10

```
# sip client
implement Sip;

Mod : con "sip";
Version : con "SIP/2.0";
Transport := "UDP";

include "sys.m";
sys: Sys;
stderr : ref Sys->FD;

include "draw.m";

include "daytime.m";

include "csget.m";

daytime: Daytime;

include "rand.m";
r : Rand;

include "kill.m";
kp : Kill;

Sip : module
{
        init : fn(ctxt: ref Draw->Context, argv: list of string);
};

# Optional audio driver for ephone - update namespace:
# bind -a /prod/shanip/module /module
include "UCBAudio.m";
ua : UCBAudio;

# Optional environment for sip setup - update namespace:
# bind -a /prod/sds/module /module
include "util/env.m";

# Ephone environment variables
home : con "/usr/inferno/config/";
eenv : con "etherenv";

# Default init values
default_ninc : con 10;
default_lport : con "5060";
default_rtpport : con "3456";
default_rrtport : con string (int default_rtpport + default_ninc);
default_client : con "8089:8089";
default_aproto : con "RTP/AVP";
default_exptime : con "3600";

# RTP and remote RTP ports (default RTCP is 1+)
Rtpport := default_rtpport;
Rrtport := default_rrtport;
Incport := default_ninc;

Gstate := (0, 0);
genrtp(client : string) : (int, int)
{
        if (client == nil) return Gstate = (0, 0);
        if (start("rand", Rtpport) || start("rand", Rrtport)) {
                Rtpport = string (int default_rtpport + random(9999, client));
                Rrtport = string (int Rtpport + Incport);
        }
        (n, v) := Gstate;
        r1 := int Rtpport + v;
        r2 := int Rrtport + v;
        inc := 2;
        if (Incport < 0) inc = -inc;
        if (++n >= (int Incport/inc)) {v += Incport; Gstate = (0, v);}
        else {v += inc; Gstate = (n, v);}
        if (Dbg) sys->print(Mod+": rtp/rrtp = %d/%d\n", r1, r2);
        return (r1, r2);
}

# Registration expiration
Exptime := default_exptime;

# Proxy/Registrar definition example:
```

```
# Proxy : string = "135.1.89.127:5060";
Proxy, Registrar : string;

# Audio protocol selection
Aproto := default_aproto;

# This client local address (or substituted address)
Laddr : string;

# Dial plan ("-dp" option)
Dialplan : string;

# Multicall mode
Multicall := 0;

active := 0;
Epid := 0;

# To reset the client process
Args : list of string;

# To delay first registration
Zreg := 0;

# Debug level
Dbg := 0;
# Verbose level -- message sent/received contents only
Vbs := 0;

init(ctxt : ref Draw->Context, args : list of string)
{
        Args = args;
        sys = load Sys Sys->PATH;
        stderr = sys->fildes(2);
        daytime = load Daytime Daytime->PATH;
        if(daytime == nil) {
          sys->fprint(stderr, Mod+": load %s: %r\n", Daytime->PATH);
          return;
        }

  if (args != nil)
    args = tl args;

        ok : int;
        user, client : string;
        args = readenv(ctxt, args);
        (ok, user, client, args) = parseopt(args);

        if (!ok) return;

        if (Laddr == nil) {
                cs := load CsGet CsGet->PATH;
                (nil, Laddr, nil) = cs->hostinfo(nil);
                if (Laddr == nil) return;
                if (Dbg) sys->print(Mod+": local address: %s\n", Laddr);
        }

        if (client == nil) client = default_client;

        if (numberp(client)) client += "@*:"+client;
        client = thisclient(client);
        sys->print(Mod+": this client: %s\n", client);

        C = ref Calls(nil, nil, nil);

        if (args != nil)
                clients = args;
        else if (Registrar == nil) {
                sys->print(Mod+": using Styx locator for SIP clients\n");
                readclients();
                registerclient(client);
        }

        ch := chan of int;
        spawn sound(ch);
        Spid = <- ch;
        if (Dbg) sys->print(Mod+": sound process %d\n", Spid);
        sip_client := user+"<sip:"+client_nonet(client)+">";
        spawn rcmd(ctxt, sip_client, ch);
        pid := <- ch;
```

```
        if (Dbg) sys->print(Mod+": command process %d\n", pid);
        sip_client = user+"<sip:"+client+">";
        sipstack(sip_client, ch);
}

initua()
{
        if (ua == nil)
                ua = load UCBAudio UCBAudio->PATH;
        if (ua != nil) {
                (ok, reason) := ua->initialize();
                if (ok < 0) {
                        sys->fprint(stderr, Mod+": %s\n", reason);
                        ua = nil;
                }
        }
}

sipstack(sip_client : string, ch : chan of int)
{
        if (!siplisten(sip_client)) return;

        if (Registrar != nil && !Zreg) {
                (user, client) := sipurlvals(sip_client);
                C.this = register(user, client, nil);
        }

        initua();

        if (ua != nil) {
                spawn listenkeys(sip_client, ch);
                Epid = <- ch;
                if (Dbg) sys->print(Mod+": ephone process %d\n", Epid);
        }
}

usage(s : string)
{
        if (s != nil) sys->print(Mod+": unknown option: %s\n", s);
        sys->print("usage: sip\t[options] [this_client] [other_client#1]... [other_client#n]\n\top
        sys->print("\n\t--\t-- include /usr/inferno/config/etherenv parameters\n\t-?\t-- this mess
        sys->print("\twrite (echo) 'help' or '?' to "+mp+"/"+sipsrv+"\t for a list of setup option
}

# read args from etherenv file on client if exists
readenv(ctxt : ref Draw->Context, args : list of string) : list of string
{
        margs : list of string;
        envp := 0;
        # -- arg is used to force reading of environment args
        for(l := args; l != nil; l = tl l)
                if (hd l == "--") {envp++; break;}
                else margs = hd l :: margs;
        if (envp) margs = reverse(margs);
        else {
                for(l := args; l != nil; l = tl l)
                        if (!findl(hd l, "-c" :: "-d" :: "-v" :: nil)) return args;
                margs = args;
        }
        args = nil;
        en := load Env Env->PATH;
        if (en == nil) sys->fprint(stderr, Mod+": %s %r\n", Env->PATH);
        else {
                en->init(ctxt, nil);
                if (Dbg) sys->print(Mod+": reading arguments from %s\n", home+eenv);
                (env, n) := en->readfenv(home+eenv);
                if (env != nil) {
                        if (Dbg) sys->print(Mod+": using env %s\n", home+eenv);
                        user := en->getenv("USER", env);
                        addr := en->getenv("IPDEV", env);
                        num := en->getenv("LINE", env);
                        loc := en->getenv("LOCATION", env);
                        protocol := en->getenv("PROTOCOL", env);
                        port := en->getenv("PORT", env);
                        proxy := en->getenv("IPLSS", env);
                        regis := en->getenv("REGISTRAR", env);
                        if (num == nil) return nil;
                        if (port == nil) port = default_lport;
                        if (protocol != nil && protocol != "udp") protocol = "tcp";
```

```
                        else protocol = "udp";
                        laddr := "*";
                        if (addr != nil) laddr = addr;
                        client := num+"@"+loc+"@"+laddr+":"+protocol+"/"+port;
                        args = client :: nil;
                        if (addr != nil) args = "-l" :: addr :: args;
                        if (user != nil) args = "-u" :: user :: args;
                        net := protocol;
                        if (regis != nil) {
                                rport : string;
                                (regis, rport) = expand2t(regis, ":");
                                if (rport == nil) rport = default_lport;
                                else {
                                        (net, rport) = expand2t(rport, "/");
                                        if (rport == nil) {rport = net; net = protocol;}
                                }
                                args = "-r" :: regis+":"+net+"/"+rport :: args;
                        }
                        if (proxy != nil) {
                                pport : string;
                                (proxy, pport) = expand2t(proxy, ":");
                                if (pport == nil) pport = default_lport;
                                else {
                                        (net, pport) = expand2t(pport, "/");
                                        if (pport == nil) {pport = net; net = protocol;}
                                }
                                args = "-p" :: proxy+":"+net+"/"+pport :: args;
                        }
                }
        }
        return append(margs, args);
}

parseopt(args : list of string) : (int, string, string, list of string)
{
        user, client : string;
        atcp := 0;
        usp := 0;
 out:
        for (; args != nil; args = tl args) {
                opt := hd args;
                case opt {
                        "?" or "-?" or "help" or "-help" or "--help" => usage(nil); return (0, nil
                        "-a" => {
                                if ((args = tl args) != nil) {
                                        case hd args {
                                                "tcp" or "TCP" or "RTP/TCP" or "RTP/TCP/AVP" => Ap:
                                                * => Aproto = default_aproto;
                                        }
                                }
                                if (Dbg) sys->print(Mod+": audio protocol set to %s\n", Aproto);
                        }
                        "-b" => {
                                if ((args = tl args) != nil) {
                                        (rt, rr) := expand2t(hd args, "/");
                                        if (rt != nil) {
                                                Rtpport = rt;
                                                if (rr != nil) Rrtport = rr;
                                                else Rrtport = string (int rt + default_ninc);
                                        }
                                        else {
                                                Rtpport = default_rtpport;
                                                Rrtport = default_rrtport;
                                        }
                                        Incport = int Rrtport - int Rtpport;
                                }
                                if (Dbg) sys->print(Mod+": audio RTP ports set to %s/%s\n", Rtpport
                        }
                        "-c" =>
                                if (Compact = !Compact) sys->print(Mod+": using compact messages\n
                                else sys->print(Mod+": using expanded keywords in messages\n");
                        "-d" => Dbg++;
                        "-dp" => {
                                if ((args = tl args) != nil) Dialplan = hd args;
                                if (Dbg) sys->print(Mod+": dial plan set to %s\n", Dialplan);
                        }
                        "-v" => Vbs++;
                        "-l" => {
                                laddr := Laddr;
                                if ((args = tl args) != nil) Laddr = hd args;
```

```
                        sys->print(Mod+": local address set %s -> %s\n", laddr, Laddr);
                }
                "-m" =>
                        if (Multicall = !Multicall) sys->print(Mod+": multiple call mode\n
                        else sys->print(Mod+": single call mode\n");
                "-p" => {
                        if ((args = tl args) != nil) {
                                val := hd args;
                                if (val == "-") {
                                        proxies := readlist("/services/server/sip_proxies"
                                        if (proxies != nil) Proxy = Registrar = hd proxies
                                        else sys->fprint(stderr, Mod+": empty /services/se
                                }
                                else {
                                        Proxy = val;
                                        if (Registrar == nil) Registrar = val;
                                }
                                if (Dbg) sys->print(Mod+": proxy set to %s\n", Proxy);
                        }
                }
                "-o" or "-o1" => {
                        if ((args = tl args) != nil) Port_offset = int hd args;
                        if (Dbg) sys->print(Mod+": port offset set to %d\n", Port_offset);
                }
                "-oa" or "-o2" => {
                        if ((args = tl args) != nil) Aport_offset = int hd args;
                        if (Dbg) sys->print(Mod+": port offset to announce %d\n", Aport_of
                }
                "-r" => {
                        if ((args = tl args) != nil) Registrar = hd args;
                        if (Dbg) sys->print(Mod+": registrar set to %s\n", Registrar);
                }
                "-t" =>
                        if (Twinport = !Twinport) sys->print(Mod+": twin port mode -- reus
                        else sys->print(Mod+": twin port is off\n");
                "-td" => {
                        if ((args = tl args) != nil) Talkdelay = int hd args;
                        if (Dbg) sys->print(Mod+": talk delay called %d frames\n", Talkdel
                }
                "-u" => {
                        if ((args = tl args) != nil) user = hd args;
                        if (Dbg) sys->print(Mod+": user is %s\n", user);
                }
                "-z" => {
                        Zreg ++;
                        if (Dbg) sys->print(Mod+": registration postponed\n");
                }
                * => {
                        if (opt != nil) {
                                if (opt[0] == '$')
                                        client = nth(int opt[1:], readlist("/services/conf
                                else if (opt[0] == '-') {if (!usp) usage(opt); usp++; cont
                                else if (client == nil) client = opt;
                                if (Dbg) sys->print(Mod+": client set to %s\n", client);
                                args = tl args;
                        }
                        break out;
                }
        }
    }
    if (atcp) tcpaudio();
    return (1, user, client, args);
}

siplisten(sip_client : string) : int
{
    ch := chan of int;
    ok : int; conn : Sys->Connection;
    (user, client) := sipurlvals(sip_client);
    if (client != nil) {
        (nil, nil, port, nil) := expandnet(client);
        net := downcase(Transport);
        (ok, conn) = announce(net, "*", port);
        if (ok < 0) return 0;
        spawn listen(client, conn, ch);
        active = <- ch;
        if (Dbg) sys->print(Mod+": listen process %d\n", active);
        if (!Monpid) {
                spawn monitor(client, ch);
                Monpid = <- ch;
```

```
                        if (Dbg) sys->print(Mod+": monitor process %d\n", Monpid);
                }
        }
        return 1;
}

Monpid := 0;
monitor(client : string, ch : chan of int)
{
        ch <- = sys->pctl(0, nil);
        while(Monpid) {
                sys->sleep(timeout);
                if (!Monpid) return;
                for (l := C.clist; l != nil; l = tl l) {
                        c := hd l;
                        if (c == nil) continue;
                        (n, e) := c.expire;
                        if (!e) continue;
                        if (n >= Retx) {
                                if (Dbg) sys->print(Mod+": max retransmit reached\n");
                                c.expire = (0, 0);
                                c.msg = nil;
                                # case where bye does not receive an ack
                                if (c.endp()) C.rem(c);
                                else c.terminate(client);
                                continue;
                        }
                        if (e <= time()) {
                                # we do resend bye
                                #if (!c.endp()) c.resend(client);
                                c.resend(client);
                                c.expire = etime(n, e);
                        }
                }
        }
}

callstatus()
{
        if ((c := C.this) != nil && C.clist != nil) c.status(2); else status("s");
}

random(range : int, client : string) : int
{
        if (r == nil) {
                r = load Rand Rand->PATH;
                if (r != nil) r->init(addnums(client)+ntime());
        }
        if (r == nil) return 1;
        else return r->rand(range);
}

kill(pid : int)
{
        if (kp == nil) kp = load Kill Kill->PATH;
        kp->killpid(string pid, array of byte "kill");
}

cleanup()
{
        genrtp(nil);
        killsound();
        pid := Monpid;
        Monpid = 0;
        sys->sleep(100);
        kill(pid);
        for (l := C.clist; l != nil; l = tl l) {
                c := hd l;
                if (c != nil && c.session != nil)
                        c.session.endaudio();
        }
        if (pid = active) {
                active = 0;
                sys->sleep(100);
                kill(pid);
        }
        if (pid = Epid) {
                Epid = 0;
                sys->sleep(100);
                kill(pid);
```

```
        }
        cleanClist(1);
        Dbg = 0;
        Proxyp = "nil";
        Toa = nil;
        Dialplan = nil;
        Twinport = 1;
        Talkdelay = 0;
        Multicall = 0;
        Rl = nil;
        if (ua != nil) {
                # ua->audioClose(0);
                sys->unmount("#a", mp);
                ua = nil;
        }
        daytime = nil;
}

# /dev/sip channel to control client from another program
# this does not deal with digit collection yet...

sipsrv : con "sip";
mp : con "/dev";

rcmd(ctxt : ref Draw->Context, sip_client : string, rch : chan of int)
{
        sys->bind("#s", mp, sys->MBEFORE);
        ch := sys->file2chan(mp, sipsrv);
        if (ch == nil) {
                rch <- = 0;
                sys->fprint(stderr, Mod+": file2chan %s/%s %r\n", mp, sipsrv);
                return;
        }
        else rch <- = sys->pctl(0, nil);

        if (Dbg) sys->print(Mod+": %s/%s is the command interpreter\n", mp, sipsrv);

        run := 1;
        reset := 0;
        mon := 0;
        while (run) {
                alt {
                        (o, data, fid, wc) := <- ch.write =>
                                if (data != nil && wc != nil) {
                                        sdata := string data;
                                        if (Dbg) sys->print(Mod+"> %s", sdata);
                                        case (s := trimspace(sdata)) {
                                                "reset" => {reset = 1; run = 0;}
                                                "status" => {mon = runstatus(mon); addstat(0, 0, f:
                                                "stopstatus" => {if (mon) mon = killstatus(mon);}
                                                * => {
                                                        if (s != "l" && s != "f") status(s);
                                                        run = sipdo(sip_client, s);
                                                }
                                        }
                                        wc <- = (len data, nil);
                                }
                        (o, n, fid, rc) := <- ch.read => {
                                err := "sip commands - write 'help' for menu";
                                if (rc != nil && n > 0) {
                                        (nil, nil, f, nil) := getstat(fid);
                                        if (f) {
                                                mon = runstatus(mon);
                                                addstat(o, n, fid, rc);
                                        }
                                        else respond(err, o, n, fid, rc, nil);
                                }
                                else if (rc != nil) rc <- = (nil, "");
                        }
                }
        }
        mon = killstatus(mon);
        cleanup();
        sys->unmount("#s", mp);
        if (reset && Dbg) sys->print(Mod+": restarting existing sip client\n");
        if (reset) spawn init(ctxt, Args);
}

respond(s : string, o, n, fid : int, rc : chan of (array of byte, string), ch : chan of int)
{
```

```
        if (ch != nil) ch <- = sys->pctl(0, nil);
        if (rc != nil && n > 0) {
                data := array of byte s;
                if (n < len data) data = data[0:n];
                rc <- = (data, "");
        }
        else if (rc != nil) rc <- = (nil, "");
}

# Optional status response feature on sip channel

runstatus(mon : int) : int
{
        if (!mon) {
                spawn statmon(ch := chan of int);
                mon = <- ch;
        }
        return mon;
}

Status : chan of string;
status(s : string)
{
        case s {
                "A" => s = "ACCEPT CALL";
                "a" => s = "DIALING...";
#               "f" => s = "FLASH";
#               "l" => s = "CALLS";
                "q" => s = "QUIT";
                "r" => s = "REGISTER";
                "z" => if (C.this != nil && C.clist != nil) s = "TERMINATE CALL"; else s = nil;
                * => if (start("a ", s)) s = "CALL"+s[1:];
        }
        if (Status != nil && s != nil) Status <- = s;
}

killstatus(mon : int) : int
{
        sys->sleep(100);
        Status = nil;
        sys->sleep(100);
        kill(mon);
        if (Dbg) sys->print(Mod+": killed statmon %d\n", mon);
        sys->sleep(100);
        tkills(Monpids, 600, 200);
        return 0;
}

statmon(ch : chan of int)
{
        Status = chan of string;
        ch <- = sys->pctl(0, nil);
        if (Dbg) sys->print(Mod+": started statmon %d\n", sys->pctl(0, nil));
        while(Status != nil) {
                s := <- Status;
                sendstat(s);
        }
}

Rl : list of (int, int, int, chan of (array of byte, string));
Monpids : list of int;
sendstat(s : string)
{
        pids : list of int;
        r : list of (int, int, int, chan of (array of byte, string));
        for (l := Rl; l != nil; l = tl l) {
                (o, n, f, rc) := hd l;
                if (rc == nil) continue;
                else {
                        spawn respond(s, o, n, f, rc, ch := chan of int);
                        Monpids = pids = (<- ch) :: pids ;
                }
                r = hd l :: r;
        }
        tkills(pids, 1000, 200);
}

addstat(o, n, fid : int, rc : chan of (array of byte, string))
{
        r : list of (int, int, int, chan of (array of byte, string));
```

```
        m := 0;
        for (l := Rl; l != nil; l = tl l) {
                (nil, nil, f, nil) := hd l;
                if (f != fid) r = hd l :: r;
        }
        Rl = (o, n, fid, rc) :: r;
}

getstat(fid : int) : (int, int, int, chan of (array of byte, string))
{
        r : list of (int, int, int, chan of (array of byte, string));
        for (l := Rl; l != nil; l = tl l) {
                (nil, nil, f, rc) := hd l;
                if (f == fid) return hd l;
        }
        sys->print("0\n");
        return (0, 0, 0, nil);
}

# Kill list of procs after timeout expires
tkills(pl : list of int, tout, quantum : int)
{
        nc := tout/quantum;
        while((pl = pidups(pl)) != nil && nc-- > 0)
                sys->sleep(quantum);
        for(; pl != nil; pl = tl pl) kill(hd pl);
}

pidups(pl : list of int) : list of int
{
        r : list of int;
        for (; pl != nil; pl = tl pl)
                if (sys->open("/prog/"+string (hd pl)+"/status", sys->OREAD) != nil) r = hd pl :: r;
        return r;
}

#nativep() : int
#{
#       return sys->open("#c/sysenv", Sys->OREAD) != nil;
#}

mkargs(s : string) : list of string
{
        if (Dbg) sys->print(Mod+": mkargs(%s) ->\n", s);
        (nil, l) := sys->tokenize(s, " \t");
        return l;
}

Call : adt
{
  conn : ref Sys->Connection;
        path : ref Path;
        fname : string;
        tname : string;
        frum : string;
        tu : string;
        fag : string;
        tag : string;
  callid : string;
        cseq : string;
        state : string;
        session : ref Session;
        expire : (int, int);
        msg : string;
        inited : int;           # True when call initiated on this end
        client : ref Client;
        store : fn(c : self ref Call, c2 : ref Call);
        rewritepath : fn(c : self ref Call);
        send : fn(c : self ref Call, client : string) : ref Call;
        resend : fn(c : self ref Call, client : string);
        resendmsg : fn(c : self ref Call, client, msg : string);
        nextstate : fn(c : self ref Call, client : string);
        disconnect : fn(c : self ref Call, client, end : string) : ref Call;
        terminate : fn(c : self ref Call, client : string) : ref Call;
        addsession : fn(c : self ref Call, sid, data : string) : int;
        addedsessionp : fn(c : self ref Call) : int;
        stateinfo : fn(c : self ref Call) : (string, int, string);
        activep : fn(c : self ref Call) : int;
        endp : fn(c : self ref Call) : int;
        registerp : fn(c : self ref Call) : int;
```

```
        mktag : fn(c : self ref Call);
        nextproxy : fn(c : self ref Call) : string;
        routeproxy : fn(c : self ref Call) : string;
        listen : fn(c : self ref Call, client : string);
        status : fn(c : self ref Call, sentp : int);
        audiop2p : fn(c : self ref Call) : int;
        switchau : fn(c : self ref Call, c2 : ref Call) : int;
};

Call.stateinfo(c : self ref Call) : (string, int, string)
{
        if (c == nil) return (nil, 0, nil);
        s := c.state;
        token, num, msg : string;
        (nil, ls) := sys->tokenize(s, " \t");
        if (ls != nil)
                if (tl ls != nil) {
                        token = hd ls;
                        num = hd tl ls;
                        for(l := tl tl ls; l != nil; l = tl l) {
                                msg += hd l;
                                if (tl l != nil) msg += " ";
                        }
                }
                else token = hd ls;
        else token = c.state;
        n := 0;
        if (num != nil)
                if (numberp(num)) n = int num;
                else sys->fprint(stderr, Mod+": unexpected state %s %s\n", token, num);
        if (Dbg > 2) sys->print(Mod+": stateinfo -> (%s, %d, %s)\n", token, n, msg);
        return (token, n, msg);
}

Call.store(c1 : self ref Call, c2 : ref Call)
{
        # could do better check route, via field and proxy here instead of in send
        if (c2.conn != nil) c1.conn = c2.conn;
        if (c2.client != nil) c1.client = c2.client;
        c1.state = c2.state;
        # preserve recorded route for future requests (e.g. ack)
        if (c2.path != nil) {
                if (c1.path == nil || c2.path.record != nil) {
                        if (Dbg) sys->print(Mod+": storing new path in call (record route) %s\n", c
# assume contact not changed
                        c1.path = c2.path;
                } else if (c1.path.route == nil) {
                        if (Dbg) sys->print(Mod+": storing new path in call (route) %s\n", c1.call
                        c1.path = c2.path;
                        if (c1.path.record == nil)
                                c1.rewritepath();
                }
                else if (c2.path.via != nil) {
                        c1.path.via = c2.path.via;
                        if (Dbg) sys->print(Mod+": storing via field in call %s\n", c1.callid);
                }
        }
        c1.frum = c2.frum;
        c1.tu = c2.tu;
        c1.inited = c2.inited;
        c1.cseq = c2.cseq;
        # multiple tags in one callid are not handled
        if (c2.fag != nil) c1.fag = c2.fag;
        if (c2.tag != nil) c1.tag = c2.tag;
}

# we assume this code is only used upon receiving response with a route
# and no record route (client does not write record-route)
Call.rewritepath(c : self ref Call)
{
        route := c.path.route;
        cont := c.path.contact;
        if (cont != nil) {
                cont = mksipurl(sipurlval(cont));
                if (Dbg) sys->print(Mod+": rewriting route in call %s using %s\n", c.callid, cont)
                r := cont :: nil;
                for (; route != nil; route = tl route)
                        if (tl route != nil) r = hd route :: r;
                c.path.route = r;
        }
```

```
        via := c.path.via;
        if (via != nil && cont != nil) {
                c.path.contact = cont;
        }
}

update_hd_route(r : list of string, client : string) : list of string
{
        s1 := sipurlval(hd r);
        p1 := find("maddr=", s1);
        if (p1 > 0) {
                dest := trimspace(s1[0:p1]);
                p2 := poso('>', s1, p1);
                if (p2 < 0) p2 = len s1;
                (l0, a0, nil) := expand(client);
                if (pos('@', l0) < 0)
                        if (Ldomain != nil) l0 += "@"+Ldomain;
                        else l0 += "@"+a0;
                if (l0 != dest) {
                        s1 = l0+";"+s1[p1:p2];
                        s1 = mksipurl(s1);
                        r = s1 :: tl r;
                        if (Dbg) sys->print(Mod+": updated first of route to %s\n", s1);
                }
        }
        return r;
}

# using tcp we can get responses on existing connection
Call.listen(c : self ref Call, client : string)
{
        if (Transport == "TCP" && c.conn != nil && c.conn.dfd != nil) {
                cl := c.client;
                if (cl != nil) {
                        spawn cl.kill(1000);
                        c.client = nil;
                }
                c.client = cl = ref Client(client, 0, 0, 0);
                spawn cl.listen(c.conn, nch := chan of int);
                pid := <- nch;
                sys->print(Mod+": tcp listener %d\n", pid);
        }
}

Call.switchau(c : self ref Call, c2 : ref Call) : int
{
        r := 0;
        if (!Multicall) return r;
        if (c2 != nil) {
                a, a2 : ref Audio;
                if (c2.session != nil) a2 = c2.session.audio;
                if (c != c2 && c != nil) {
                        if (c.session != nil) a = c.session.audio;
                        if (a != nil && a2 != nil) {
                                (lch, sch) := a.lchs;
                                if (lch == nil || sch == nil) a.lchs = (chan of int, chan of int);
                        }
                }
                if (a2 != nil) {
                        if (c == nil)
                                for (l := C.clist; l != nil; l = tl l) {
                                        c = hd l;
                                        if (c == c2 || c.session == nil) continue;
                                        if ((a = c.session.audio) != nil) {
                                                (lch, sch) := a.lchs;
                                                if (lch == nil || sch == nil) a.lchs = (chan of int
                                        }
                                }
                        (lch, sch) := a2.lchs;
                        a2.lchs = (nil, nil);
                        if (lch != nil) lch <- = a2.listen;
                        if (sch != nil) sch <- = a2.speak;
                        r = lch != nil && sch != nil;
                        if (Dbg) sys->print(Mod+": audio channel switched = %d\n", r);
                }
        }
        return r;
}

Session : adt
```

```
{
  sid   : string;
        data : string;
        rdata : list of string;
        audio : ref Audio;
        endaudio : fn(s : self ref Session);
        startaudio : fn(s : self ref Session, tipe : int);
        dialaudio : fn(s : self ref Session);
        announceaudio : fn(s : self ref Session);
};

Audio : adt
{
        addr1 : string;
        addr2 : string;
        # tipe = 1 when call received by this UA
        tipe : int;
        conn1 : ref Sys->Connection;
        conn2 : ref Sys->Connection;
        listen : int;
        speak : int;
        rtcp1 : int;
        rtcp2 : int;
        ccon1 : ref Sys->Connection;
        ccon2 : ref Sys->Connection;
        size : int;
        # value set to frame size of the first audio received
        busy : int;
        # (listen, speak) lock channels used to turn on/off each audio channel path
        lchs : (chan of int, chan of int);
        # true when talking point to point (not to a resource server)
        p2p : int;
};

Calls : adt
{
        clist : list of ref Call;
        this : ref Call;
        recv : ref Call;
        find : fn(cl : self ref Calls, id : string) : ref Call;
        finda : fn(cl : self ref Calls, a : ref Audio) : ref Call;
        item : fn(cl : self ref Calls, n : int) : ref Call;
        next : fn(cl : self ref Calls) : ref Call;
        take : fn(cl : self ref Calls, c : ref Call);
        add : fn(cl : self ref Calls, c : ref Call);
        rem : fn(cl : self ref Calls, c : ref Call) : int;
        remrecv : fn(cl : self ref Calls, c : ref Call) : int;
        print : fn(cl : self ref Calls);
        soleaup : fn(cl : self ref Calls, s : ref Session) : int;
        multiaup : fn(cl : self ref Calls) : int;
};

# Master call list
C : ref Calls;

Calls.print(cl : self ref Calls)
{
        sys->print(Mod+": call list:\n");
        i := 0;
        ct := C.this;
        cr := C.recv;
        r : string;
        for (l := cl.clist; l != nil; l = tl l) {
                c := hd l;
                mode : string;
                if (ct == c) {mode = "<-"; ct = nil;}
                else if (C.recv == c) {mode = "recv call"; cr = nil;}
                else mode = "";
                r += sys->sprint("%d: %s %s %s\n", ++i, c.callid, c.state, mode);
        }
        sys->print("%s", r);
        status("CALLS: "+r);
        if (ct != nil) sys->print("?: idle %s %s this call\n", ct.callid, ct.state);
        if (cr != nil) sys->print("?: idle %s %s recv call\n", cr.callid, cr.state);
}

Calls.find(cl : self ref Calls, id : string) : ref Call
{
        for (l := cl.clist; l != nil; l = tl l)
          if ((hd l).callid == id) return hd l;
```

```
        return nil;
}

Calls.finda(cl : self ref Calls, a : ref Audio) : ref Call
{
        if (a == nil) return nil;
        for (l := cl.clist; l != nil; l = tl l)
          if ((s := (hd l).session) != nil && s.audio == a) return hd l;
        return nil;
}

Calls.item(cl : self ref Calls, n : int) : ref Call
{
        i := 1;
        for (l := cl.clist; l != nil; l = tl l)
                if (i == n) return hd l;
                else i++;
        return nil;
}

Calls.next(cl : self ref Calls) : ref Call
{
        c := cl.this;
        for (l := cl.clist; l != nil; l = tl l)
                if (hd l == c)
                        if (tl l != nil) return hd tl l;
                        else return hd cl.clist;
        return nil;
}

Calls.take(cl : self ref Calls, c : ref Call)
{
        if (c != nil) {
                swc : ref Call;
                if (cl.clist != nil && cl.this != nil && cl.this.callid != c.callid) {
                        swc = cl.this;
                        if (Proxy != nil && c.conn == nil && cl.this.conn != nil) c.conn = cl.this
                        if (Dbg > 1) sys->print(Mod+": switching call %s -> %s\n", cl.this.callid,
                }
                pc := cl.find(c.callid);
                if (pc != nil) {
                        if (pc == c) {
                                cl.this = c;
                                cl.remrecv(c);
                                if (swc != nil) swc.switchau(c);
                                return;
                        }
                        else cl.rem(pc);
                }
                cl.clist = (cl.this = c) :: cl.clist;
                if (swc != nil) swc.switchau(c);
        }
}

Calls.add(cl : self ref Calls, c : ref Call)
{
        if (c != nil) {
                pc := cl.find(c.callid);
                if (pc != nil) cl.rem(pc);
                cl.clist = c :: cl.clist;
        }
}

Calls.remrecv(cl : self ref Calls, c : ref Call) : int
{
        if (cl.recv != nil && cl.recv.callid == c.callid)       {
                cl.recv = nil;
                return 1;
        }
        return 0;
}

Calls.rem(cl : self ref Calls, c : ref Call) : int
{
        if (c == nil) return 0;
        if (Dbg > 1) sys->print(Mod+": removing call %s\n", c.callid);
        n := 0;
        r : list of ref Call;
        for (l := cl.clist; l != nil; l = tl l)
                if (hd l != c && (hd l).callid != c.callid) r = hd l :: r;
```

```
                else n++;
        cl.clist = reversec(r);
        # reset port count
        if (cl.clist == nil) genrtp(nil);
        if (cl.this == c) {
                cl.this = nil;
                for (l := cl.clist; l != nil; l = tl l)
                        if (!(hd cl.clist).registerp()) {
                                cl.this = hd cl.clist;
                                break;
                        }
        }
        if (cl.recv == c) cl.recv = nil;
        cl.remrecv(c);
        # add status update
        callstatus();
        # may need to turn audio channel on
        c.switchau(cl.this);
        return n;
}

# there is no other call with audio besides the one owning this session
Calls.soleaup(cl : self ref Calls, s : ref Session) : int
{
        if (!Multicall)
                for (l := cl.clist; l != nil; l = tl l) {
                        c := hd l;
                        if (c.session != nil && c.session != s && c.session.audio != nil) return 0
                }
        return 1;
}

# there is more than one call present and one has audio
Calls.multiaup(cl : self ref Calls) : int
{
        n := 0;
        if (Multicall && C.clist != nil && tl C.clist != nil)
                for (l := cl.clist; l != nil; l = tl l) {
                        c := hd l;
                        if (c.session != nil && c.session.audio != nil) {n++; break;}
                }
        if (n && Dbg) sys->print(Mod+": multiple audio calls\n");
        return n;
}

reversec(l : list of ref Call) : list of ref Call
{
        r : list of ref Call;
        for(; l != nil; l = tl l) r = hd l :: r;
        return r;
}

itselfp(cs, client : string) : int
{
        if (Dbg) sys->print(Mod+": calling %s from %s\n", cs, client);
        (l, a, p, nil) := expandnet(client);
        if (pos('@', l) < 0) return l+"@"+a == cs;
        else return l == cs;
        return 0;
}

sipdo(sip_client, cmd : string) : int
{
        c := C.this;
        (nil, cl) := sys->tokenize(cmd, " \t\r\n");
        if (cl == nil) return 1;
        Sch <- = ("", 0);
        (user, client) := sipurlvals(sip_client);
        case hd cl {
                "a" or "A" => {
                        if (tl cl != nil) {
                                line := hd tl cl;
                                (tname, tu) := sipurlvals(line);
                                if (tu != nil) line = tu;
                                called : string;
                                if (Proxy == nil) {
                                        if (Ldomain != nil && pos('@', line) < 0)
                                                called = findclient(line+Ldomain);
                                        if (called == nil)
                                                called = findclient(line);
```

```
                }
                else if (Ldomain != nil && pos('@', line) < 0) called = line + Ldo
                else called = line;

                if (itselfp(called, client)) {
                        sys->fprint(stderr, Mod+": calling self %s\n", client);
                        Sch <- = ("b", -1);
                }
                else if (called != nil) {
                        if (Dbg) sys->print(Mod+": calling %s\n", called);
                        c = connect(user, tname, client, called, c);
                        C.take(c);
                        Sch <- = ("", 0);
                }
                else {
                        sys->fprint(stderr, Mod+": client not found at line %s\n",
                        Sch <- = ("x", -1);
                }
        }
        else if (c != nil && !c.registerp()) {
                if (answercall(c, client)) return 1;
                if (start("INVITE ", c.state))
                c.nextstate(client);
                else
                if (Dbg) sys->print(Mod+": in call %s %s\n", c.callid, c.state);
        }
        else {
                sys->fprint(stderr, Mod+": missing line number\n");
                Sch <- = ("x", -1);
        }
        return 1;
}
"f" => {
        if (tl cl != nil) {
                cn := int hd tl cl;
                c = C.item(cn);
                if (c != nil) {C.take(c); c.status(2);}
                else sys->fprint(stderr, Mod+": call number %d not found\n", cn);
        }
        else if (answercall(c, client)) return 1;
        else {
                c = C.next();
                if (c != nil && c != hd C.clist) {C.take(c); c.status(2);}
                else if (keycall()) status("DIALING...");
                else if (c != nil) {C.take(c); c.status(2);}
        }
        return 1;
}
"l" => {
        C.print();
        return 1;
}
"r" or "s" => {
        cc := c;
        if (hd cl == "s") cc = nil;
        c = register(user, client, cc);
        if (C.this == nil) C.this = c;
        return 1;
}
"z" => {
        if (c == nil) c = C.this = C.recv;
        if (c == nil) {
                callstatus();
                sys->fprint(stderr, Mod+": no current call\n");
        }
        else {
                (method, code, reason) := c.stateinfo();
                if ((method == "INVITE" && code < 300) || method == "ACK") {
                        ex := "BYE";
                        if (method == "INVITE" && code < 200) ex = "CANCEL";
                        c = c.disconnect(client, ex);
                        C.take(c);
                        #C.rem(c); will be removed after ack
                        # will timeout if no ack received
                        if (ex == "BYE") c.expire = etime(0, 0);
                        return 1;
                }
                else {
                        if (method != "BYE" && method != "REGISTER") {
                                c = c.disconnect(client, "CANCEL");
```

```
                                        C.take(c);
                                }
                                C.rem(c);
                                return 1;
                        }
                }
                S.s = S.d = nil;
                if (c != nil) {
                        c.inited = 0;
                        C.rem(c);
                }
                cleanClist(0);
        }
        "q" => return 0;
        * => {
                cmd := hd cl;
                if (cmd != nil) {
                        case cmd[0] {
                                '-' or '=' => {
                                        eql := cmd[0] == '=';
                                        cl = tl cl;
                                        if ((cmd = cmd[1:]) != nil) cl = cmd :: cl;
                                        if (eql) cl = "=" :: cl;
                                        test(cl);
                                        return 1;
                                }
                        }
                }
                usage_call();
        }
    }
    return 1;
}

answercall(c : ref Call, client : string) : int
{
    if (c.state == "INVITE 180 Ringing") {
        c.state = "INVITE 200 OK";
        c.send(client);
        return 1;
    }
    return 0;
}

usage_call()
{
    sys->fprint(stderr, Mod+": a <number>, f, l, q, r, s, z, and (-, =)[cmd] : are supported c
}

include "qidcmp.m";
qc : Qidcmp;
Cdir : import qc;

clients : list of string;

Spath : con "/services/server/sip_clients";
Sqid : ref Cdir;

# load the last record of all clients that connected via sip
readclients() : list of string
{
    if (Sqid == nil) {
        qc = load Qidcmp Qidcmp->PATH;
        if (qc == nil) {
            sys->fprint(stderr, Mod+": %s %r\n", Qidcmp->PATH);
            return nil;
        }
        qc->init(nil, nil);
        Sqid = ref Cdir(nil, Qidcmp->SAME);
    }
    if (Sqid.fcmp(Spath)) {
        if (Dbg) sys->print(Mod+": updating Styx SIP clients\n");
        clients = readlist(Spath);
    }
    return clients;
}

addclients(client : string)
{
    if (Dbg) sys->print(Mod+": adding SIP client locator %s\n", client);
```

```
        fappend(Spath, client);
        readclients();
}

replaceclients(new, old : string)
{
        if (Dbg) sys->print(Mod+": updating SIP client locator %s -> %s\n", old, new);
        r : list of string;
        for (l := clients; l != nil; l = tl l)
                if (hd l == old) r = new :: r;
                else r = hd l :: r;
        clients = reverse(r);
        writelist(Spath, clients);
        readclients();
}

registerclient(client : string)
{
        (l, a, p) := expand(client);
        host := findclient(l);
        if (host == nil)
                addclients(client);
        else if (host != client)
                replaceclients(client, host);
}

findclient(line : string) : string
{
        readclients();
        for(l := clients; l != nil; l = tl l) {
                (num, nil, nil) := expand(hd l);
                if (num == line) return hd l;
        }
        return nil;
}

# Local domain starts with @
Ldomain : string;

thisclient(client : string) : string
{
        (who, laddr, port) := expand(client);
        if ((p := pos('@', who)) >= 0) {
                Ldomain = who[p:];
                if (Dbg) sys->print(Mod+": local domain %s\n", Ldomain);
        }
        if (Laddr != laddr && addressp(laddr) == 1) {
                sys->print(Mod+": local address substitute: %s -> %s\n", Laddr, laddr);
                Laddr = laddr;
        }
        return who+"@"+laddr+":"+thisport(port);
}

thisport(p : string) : string
{
        (n, lp) := sys->tokenize(p, "/");
        case n {
                1 => Transport = "UDP";
                2 => {Transport = upcase(hd lp); p = hd tl lp;}
                * => sys->fprint(stderr, Mod+": unexpected port argument %s\n", p);
        }
        return downcase(Transport)+"/"+p;
}

ntime() : int
{
        return int 1e+09 + daytime->now();
}

rtime() : int
{
        return daytime->now();
}

# Retransmission algorithm
timeout := 200;
Timeout := 5000;
Toa : array of int;
Retx : con 7;
```

```
mktoa()
{
        if (Toa == nil) {
                Toa = array[Retx] of int;
                Toa[0] = timeout;
                if (Dbg || Vbs) {
                        sys->print(Mod+": set retransmission times (%d (really 1000), %d)\n", time
                        # Debugging slow down
                        if (Toa[0] < 1000) Toa[0] = 1000;
                }
                for (i := 1; i < len Toa; i++) {
                        t := Toa[i -1]; t += t;
                        if (t > Timeout) Toa[i] = Timeout;
                        else Toa[i] = t;
                }
        }
}

etime(n, t : int) : (int, int)
{
        mktoa();
        if (n < 0) n = 0;
        if (n >= Retx) return (n, t);
        if (t == 0) t = time();
        return (n+1, t + Toa[n]);
}

time() : int
{
        return sys->millisec();
}

explicitport(entry, port : string) : string
{
        if (port != default_lport) entry += ":"+port;
        else if (Proxy != nil) {
                (nil, nil, pp, nil) := expandnet(Proxy);
                if (port != pp) entry += ":"+port;
        }
        if (Dbg > 2) sys->print(Mod+": explicitport() -> %s\n", entry);
        return entry;
}

proxy(client : string) : string
{
        if (Proxy == nil) return client;
        return Proxy;
}

proxytype() : string
{
        if (Proxy != nil) {
                (t, nil, nil) := expand(Proxy);
                return t;
        }
        return nil;
}

mkvia(client : string) : string
{
        if (client != nil) {
                (nil, vaddr, vport, net) := expandnet(client);
                # remove the line@ since it crashes vovida phones
                #return " "+Version+"/"+upcase(net)+" "+client;
                entry := " "+Version+"/"+upcase(net)+" "+vaddr;
                return explicitport(entry, vport);
        }
        return nil;
}

viavalnet(via : string) : (string, string)
{
        (proto, vhost) := expand2t(via, " \t");
        tp := snth_token(2, proto, "/");
        transport := Transport;
        if (tp == nil) {
                sys->fprint(stderr, Mod+": unexpected transport protocol %s in via field\n", proto
        }
        else transport = tp;
        (nil, maddr) := split(vhost, "maddr=");
```

```
        if (maddr != nil) vhost = maddr;
        return (vhost, downcase(transport));
}

viaval(via : string) : string
{
        (vhost, nil) := viavalnet(via);
        return vhost;
}

viahost(c : ref Call, default : string, rcv : int) : string
{
        vhost := default;
        r := c.routeproxy();
        if (r != nil) r = "@"+r;
        else {
                transport := downcase(Transport);
                if (c.path.via == nil) {
                        if (c.path.contact != nil) vhost = c.path.contact;
                        if (rcv)
                                sys->fprint(stderr, Mod+": error received empty via field - using '
                        else if (Dbg > 1) sys->print(Mod+": via () - using %s\n", vhost);
                }
                else {
                        (vhost, transport) = viavalnet(hd c.path.via);
                        if (Dbg > 1) sys->print(Mod+": via host %s\n", vhost);
                }
                (n, a, p) := expand(vhost);
                r = "@"+a+":"+transport+"/"+p;
        }
        if (Dbg > 1) sys->print(Mod+": viahost returns %s\n", r);
        return r;
}

Call.nextproxy(c : self ref Call) : string
{
        viaproxy := Proxy;
        via := c.path.via;
        ## the tl check is wrong but i still receive single via fields back from proxy!
        if (via != nil && tl via != nil) {
                net : string;
                (viaproxy, net) = viavalnet(hd via);
                (nil, maddr) := split(viaproxy, "maddr=");
                if (maddr != nil) viaproxy = maddr;
                port : string;
                (nil, maddr, port, nil) = expandnet(viaproxy);
                return maddr+":"+net+"/"+port;
        }
        return viaproxy;
}

Call.routeproxy(c : self ref Call) : string
{
        rproxy : string;
        route := c.path.route;
        if (route != nil && tl route != nil) {
                (nil, net) := split(hd route, "transport=");
                if (net != nil) (net, nil) = expand2t(net, "; \t");
                if (net == nil) net = downcase(Transport);
                rproxy = sipurlval(hd route);
                (nil, maddr) := split(rproxy, "maddr=");
                if (maddr != nil) rproxy = maddr;
                port : string;
                (nil, maddr, port, nil) = expandnet(rproxy);
                rproxy = maddr+":"+net+"/"+port;
        }
        return rproxy;
}

viaproxy(proxy, contact : string, via : list of string) : list of string
{
        if (proxy != nil) return mkvia(proxy) :: via;
        else if (contact != nil) return mkvia(contact) :: via;
        else return via;
}

eqproxies(x, y : string) : int
{
        (nil, m1, p1, n1) := expandnet(x);
        (nil, m2, p2, n2) := expandnet(y);
```

```
        return m1 == m2 && p1 == p2 && n1 == n2;
}

numberp(s : string) : int
{
        for (i := 0; i < len s; i++) {
                c := s[i];
                if (c < '0' || c > '9') return 0;
        }
        return 1;
}

# may be an ip address field
addrfp(s : string) : int
{
        for (i := 0; i < len s; i++) {
                c := s[i];
                if ((c >= '0' && c <= '9') || (c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z') || (
                else return 0;
        }
        return 1;
}

# return 1 if numeric address > 1 if hostname 0 if neither
addressp(s : string) : int
{
        (n, l) := sys->tokenize(s, ".");
        r := 1;
        if (n <= 1) {
                if (l != nil && addrfp(hd l)) ++r;
                else return 0;
        }
        else for (; l != nil; l = tl l)
                if (!numberp(hd l))
                        if (addrfp(hd l)) ++r;
                        else return 0;
        return r;
}

expand(client : string) : (string, string, string)
{
        (n, la) := sys->tokenize(client, "@");
        pa := client;
        line, addr, port : string;
        if (n >= 2) {
                if (n > 2)
                        for(l := la; l != nil; l = tl l) {
                                line += hd l;
                                if (tl l != nil)
                                        if (tl tl l != nil) line += "@";
                                        else { pa = hd tl l; break; }
                        }
                else {
                        line = hd la;
                        pa = hd tl la;
                }
        }
        else if (n) {line = "()"; pa = hd la;}

        (n, la) = sys->tokenize(pa, ":");
        if (n >= 2) {
                if (addressp(hd la)) addr = hd la;
                else {
                        addr = Laddr;
                        if (line == nil) line = hd la;
                }
                port = hd tl la;
        }
        else {
                if (line == nil) {
                        line = pa;
                        addr = Laddr;
                }
                else {
                        addr = pa;
                        if (numberp(addr)) addr = Laddr;
                }
                port = default_lport;
        }
        if (line == "()") line = nil;
```

```
        if (Dbg > 2) sys->print(Mod+": expand(%s) -> (%s, %s, %s)\n", client, line, addr, port);
        return (line, addr, port);
}

Call.registerp(c : self ref Call) : int
{
        if (c == nil) return 0;
        return start("REGISTER", c.state);
}

register(user, frum : string, c : ref Call) : ref Call
{
        if (Registrar == nil)
                Registrar = Proxy;
        if (Registrar == nil)
                sys->fprint(stderr, Mod+": no registar nor proxy defined\n");
        else {
                reg := Registrar;
                rconn : ref Sys->Connection;
                newp := 0;
                if (c != nil && (c.registerp() || (Proxy != nil && reg == Proxy)))
                        rconn = c.conn;
                if (rconn == nil) {
                        (vline, vaddr, vport, net) := expandnet(reg);
                        if (Dbg) sys->print(Mod+": connect to %s at %s!%s!%s\n", vport, net, vaddr
                        (ok, conn) := rdial(net, vaddr, vport, localport(frum, vport, vaddr));
                        rconn = ref conn;
                        if (ok < 0) return nil;
                        newp++;
                }
                callid := sid2callid(string ntime())+"@"+Laddr;
                path := ref Path(nil, viaproxy(Proxy, nil, mkvia(frum) :: nil), nil, nil);
                frum = client_nonet(frum);
                if (c != nil) {
                        c = ref Call(rconn, path, user, nil, frum, reg, nil, nil, callid, nil, "RE(
                        c.send(frum);
                        c.expire = (0, int Exptime);
                        if (newp) c.listen(frum);
                        return c.send(frum);
                }
                else {
                        c = ref Call(rconn, path, user, nil, frum, reg, nil, nil, callid, nil, "RE(
                        if (newp) c.listen(frum);
                        return c.send(frum);
                }
        }
        return c;
}

connect(fname, tname, frum, tu : string, c : ref Call) : ref Call
{
        rconn : ref Sys->Connection;
        if (c.registerp() && Proxy != nil && Proxy == Registrar)
                rconn = c.conn;
        if (rconn == nil) {
                (vline, vaddr, vport, net) := expandnet(proxy(tu));
                if (Dbg) sys->print(Mod+": connect to %s at %s!%s!%s\n", vport, net, vaddr, vport)
                (ok, conn) := rdial(net, vaddr, vport, localport(frum, vport, vaddr));
                if (ok < 0) return nil;
                rconn = ref conn;
        }
        callid := sid2callid(string ntime())+"@"+Laddr;
        path := ref Path(nil, viaproxy(Proxy, tu, mkvia(frum) :: nil), nil, nil);
        frum = client_nonet(frum);
        tu = client_nonet(tu);
        c = ref Call(rconn, path, fname, tname, frum, tu, mktag(), nil, callid, nil, "INVITE", nil
        c.listen(frum);
        return c.send(frum);
}

client_nonet(s : string) : string
{
        (n, a, p, nil) := expandnet(s);
        return n+"@"+a+":"+p;
}

# simplify client name (no net, no default port)
client_sname(s : string) : string
{
        (n, a, port, nil) := expandnet(s);
```

```
        s = n+"@"+a;
        if (port != default_lport) s += ":"+port;
        else if (Proxy != nil) {
                (nil, nil, pp, nil) := expandnet(Proxy);
                if (port != pp) s += ":"+port;
        }
        return s;
}

# support 4567@1.2.3.4:tcp/5566 format
expandnet(s : string) : (string, string, string, string)
{
        (n, a, p) := expand(s);
        (net, port) := netport(p);
        return (n, a, port, net);
}

# support tcp/5060 format in proxy and client definitions
netport(np : string) : (string, string)
{
        (net, p) := expand2t(np, "/");
        if (p != nil) np = p;
        else net = downcase(Transport);
        return (net, np);
}

# first port offset -- avoid port numbering collision on same host
Port_offset := 0;

# second port offset -- if port == the announced port
Aport_offset := 0;
#Aport_offset := 30000;

localport(client, port, taddr : string) : string
{
        (nil, nil, cport, nil) := expandnet(client);
        # if we already used this port -- change it
        # note: lss sip server 3.0 responds on the source port so source
        # port needs to be same as client we announced -- should not be required!
        if (port == cport) {
                # for collocated clients announced port offset is provided
                if (Aport_offset) return string (Aport_offset + int port);
                else return cport;
        }
        else if (Proxy != nil) {
                (nil, nil, pp, nil) := expandnet(Proxy);
                if (port == pp) return cport;
        }
        if (Port_offset) port = string (int port + Port_offset);
        if (taddr == Laddr) return "1"+port;
        else return port;
}

Call.disconnect(c : self ref Call, client : string, end : string) : ref Call
{
        c.state = end;
        if (c.session != nil) c.session.endaudio();
        c = c.send(client);
        c.inited = 0;
        return c;
}

Call.terminate(c : self ref Call, client : string) : ref Call
{
        c = c.disconnect(client, "BYE");
        C.take(c);
        c = c.disconnect(client, "CANCEL");
        C.take(c);
        return c;
}

# may detect point to point call when receiving ACK for lazy speak feature
Call.audiop2p(c : self ref Call) : int
{
        if (c.session != nil && (a := c.session.audio) != nil) {
                if (!a.tipe) sys->fprint(stderr, Mod+": audiop2p is not for a.tipe=%d\n", a.tipe);
                else {
                        if (c.path != nil && c.path.contact != nil) {
                                (nil, caddr, nil) := expand(sipurlval(c.path.contact));
                                (faddr, nil, nil) := expand3t(a.addr1, ":");
```

```
                                a.p2p = caddr == faddr && caddr != nil && faddr != nil;
                                if (Dbg) sys->print(Mod+": audiop2p -> %s == %s\n", faddr, caddr);
                        }
                }
                return a.p2p;
        }
        return 0;
}

Sipmethods : list of string;

sipmethodp(s : string) : int
{
        if (Sipmethods == nil) Sipmethods = "OPTIONS" :: "REGISTER" :: "INVITE" :: "ACK" :: "BYE"
        cnt := 1;
        for(l := Sipmethods; l != nil; l = tl l)
                if (start(hd l, s)) return cnt;
                else cnt++;
        return 0;
}

cseqmethodp(s : string) : int
{
        return sipmethodp(s) > 1;
}

Call.endp(c : self ref Call) : int
{
        s := c.state;
        return s == "CANCEL" || s == "BYE" || start("BYE ", s);
}

endp(method : string) : int
{
        return method == "CANCEL" || method == "BYE";
}

# Type of message sent
# 0 long form
# 1 short form
Compact := 1;

field(f : string) : string
{
        if (!Compact) return f;
        case f {
                "From" => return "f";
                "To" => return "t";
                "Call-ID" => return "i";
                "Via" => return "v";
                "Content-Encoding" => return "e";
                "Content-Length" => return "l";
                "Content-Type" => return "c";
                "Contact" => return "m";
                "Subject" => return "s";
                # discrepencies from vovida -> from LSS and 3com
                "Cseq" => return "CSeq";
        }
        return f;
}

# (Vijay substract the sum of charnums from the proxy address)
# add charsums of the local address for UAS tag
mktag() : string
{
        s := Laddr;
        nt := ntime();
        for (i := 0; i < len s; i++) nt += s[i];
        return sys->sprint("%x", nt);
}

Call.mktag(c : self ref Call)
{
        if (c.tag != nil) return;
        c.tag = mktag();
}

Proxyp := "nil";
proxyp(t : string) : int
{
```

```
        if (Proxyp == "nil") {
                proxyp := proxytype();
                if (proxyp != nil && proxyp != Proxyp && Dbg)
                        sys->print(Mod+": note: using SIP proxy type %s\n", proxyp);
                Proxyp = proxyp;
        }
        return t == Proxyp;
}

addnums(s : string) : int
{
        for ((n, i) := (0, 0); i < len s; i++) n += s[i];
        return n;
}

genseqn(client : string) : int
{
#       if (!proxyp("lss")) return 1;
        return random(99999, client);
}

Call.send(c : self ref Call, client : string) : ref Call
{
        (method, code, reason) := c.stateinfo();
        rcode := 0;
        # clean the ACK kludge
        if (method == "ACK" && code) (c.state, rcode, code, reason) = (method, code, 0, nil);
        if (!sipmethodp(method)) {
                sys->fprint(stderr, Mod+": unknown SIP event %s\n", method);
                return nil;
        }
        if (Dbg) sys->print(Mod+": current state %s %d %s\n", method, code, reason);
        frum := c.frum;
        tu := c.tu;
        callid := c.callid;

        if (c.callid == nil) sys->fprint(stderr, Mod+": missing callid in call to %s\n", tu);

        (lline, laddr, lport, nil) := expandnet(client);
        (fline, faddr, fport) := expand(frum);
        (tline, taddr, tport) := expand(tu);

        # fix first time call id (now that we preserve @)
        if (pos('@', callid) < 0) callid += "@"+faddr;

        # contact
        cont := explicitport(laddr, lport);

        orig, dest : string;
        if (pos('@', fline) >= 0) {
                orig = fline;
                (fline, nil) = expand2t(fline, "@");
        }
        else if (fline != nil)
                orig = explicitport(fline+"@"+faddr, fport);
        else orig = explicitport(faddr, fport);

        if (pos('@', tline) >= 0) {
                dest = tline;
                (tline, nil) = expand2t(tline, "@");
        }
        else if (tline != nil)
          dest = explicitport(tline+"@"+taddr, tport);
        else dest = explicitport(taddr, tport);

        header, data : string;
        # This was to talk to vovida
        addp := !c.inited;
        addp = 0;
        if (!code) {
                if (c.registerp()) addp = 0;
                # rfc2543 line 1754
                if (method == "REGISTER" && Ldomain != nil) header += method+" sip:"+Ldomain[1:];
                else header += method+" sip:"+add_lport(dest, addp);
                if (addp) header += ";user=phone";
                header += " ";
        }

        header += Version;
        if (code) header += sys->sprint(" %d %s", code, reason);
```

```
	header += "\r\n";
	record := c.path.record;
	route := c.path.route;
	routep := 0;
	if (record != nil && code >= 200 && !endp(method)) {
		header += field("Record-Route")+": ";
		for (r := record; r != nil; r = tl r) {
			header += mksipurl(hd r);
			if (tl r != nil) header += ", ";
		}
		header += "\r\n";
		if (route == nil) sys->fprint(stderr, Mod+": missing route field with record-route
	}
	else routep = route != nil;

	rproxy := c.routeproxy();
	if (routep && c.state != "CANCEL" && !(method == "BYE" && code)) {
		r := route;
		if (rproxy != nil) r = tl r;
		else r = update_hd_route(route, client);
		if (!Compact)
			for (; r != nil; r = tl r)
				header += field("Route")+": "+mksipurl(hd r)+"\r\n";
		else {
			header += field("Route")+": ";
			for (; r != nil; r = tl r) {
				header += mksipurl(hd r);
				if (tl r != nil) header += ", ";
			}
			header += "\r\n";
		}
	}

	if (rproxy == nil)
		rproxy = c.nextproxy();
	if (Proxy != nil && !eqproxies(Proxy, rproxy)) {
		if (Dbg) sys->print(Mod+": switching proxy %s -> %s\n", Proxy, rproxy);
		c.conn.dfd = nil; c.conn = nil;
	}

	via := c.path.via;
	# this is a response (route is on)
	if (method == "ACK") {
		if (Proxy != nil && len via < 2) {
			sys->fprint(stderr, Mod+": response missing via field (%s)\n", method);
			via = viaproxy(Proxy, nil, mkvia(cont) :: nil);
		}
	}
	if (via != nil && tl via != nil && method != "ACK") {
		if (!code) via = tl via;
		for (; via != nil; via = tl via)
			header += field("Via")+":"+hd via+"\r\n";
	}
	else if (via != nil && !eqproxies(viaval(hd via), Proxy))
		header += field("Via")+":"+hd via+"\r\n";
	else header += field("Via")+":"+ mkvia(cont)+"\r\n";

	# disable this now - was needed to talk to Vovida 1.7
	addp = 0;
	sipo := c.fname; sipd := c.tname;
	if (proxyp("lss")) sipo = sipd = nil;
	if (code >= 200 && method == "BYE") {
		sipo += "<sip:"+orig+">";
		sipd += "<sip:"+add_lport(dest, addp)+">";
	}
	else if (c.registerp()) {sipd += "<sip:"+orig+">"; sipo = sipd;}
	else {
		sipo += "<sip:"+add_lport(orig, addp)+">";
##		sipd += "<sip:"+add_lport(dest, addp)+";user=phone>";
		sipd += "<sip:"+add_lport(dest, addp)+">";
	}

	header += field("From")+": "+sipo;
	if (c.fag != nil) if (!proxyp("lss")) header += ";tag="+c.fag;

	if (code >= 200 && method == "INVITE") c.mktag();
	else if (method == "ACK" && c.tag == nil && !rcode) {
		c.mktag();
		sys->fprint(stderr, Mod+": missing tag for ACK - made up %s\n", c.tag);
	}
```

```
	header += "\r\n"+field("To")+": "+sipd;
	if (c.tag != nil) if (!proxyp("lss")) header += ";tag="+c.tag;

	header += "\r\n"+field("Call-ID")+": "+ callid +"\r\n";

	restart := 0;
	if (c.cseq == nil)
		c.cseq = string genseqn(client)+" "+method;
	if (endp(method)) {
		(nseq, nil) := expand2t(c.cseq, " \t");
		if (!code) nseq = string (int nseq +1);
		else if (code == 200) restart = c.inited;
		c.cseq = string nseq+" "+method;
	}
	else {
		(nseq, nil) := expand2t(c.cseq, " \t");
		if (cseqmethodp(method))
			c.cseq = nseq+" "+method;
	}
	header += field("CSeq")+": "+c.cseq+"\r\n";

	curl : string;
	if (proxyp("lss")) curl = "<sip:"+cont+">";
	else {
		if (pos('@', lline) >= 0) (lline, nil) = expand2t(lline, "@");
		curl = "<sip:"+lline+"@"+cont+">";
	}

	if (method != "REGISTER")
		header += field("Contact")+": "+curl+"\r\n";
	if (!code && method == "INVITE") {
		header += "User-Agent: Inferno Webphone 2630\r\n";
		header += field("Subject")+": Inferno Webphone INVITE\r\n";
		header += field("Content-Type")+": application/sdp\r\n";
	}
	if (code == 200 && method == "INVITE") {
		header += field("Content-Type")+": application/sdp\r\n";
	}
	if (method == "REGISTER") {
		(n, e) := c.expire;
		if (!e) header += field("Contact")+": *\r\nExpires: 0\r\n";
		else {
			header += field("Contact")+": "+curl;
			if (Transport != "UDP") header += +";transport="+downcase(Transport);
			header += "\r\nExpires: "+string e+"\r\n";
		}
		c.expire = (0, 0);
	}
	header += field("Content-Length")+": ";

	csp := 0;
	if ((!code || code == 200) && method == "INVITE") {
		(rtpport, rrtport) := genrtp(client);
		daddr := laddr; # was faddr
		if (code == 200) {
			(rrtport, rtpport) = (rtpport, rrtport);
#			daddr = derive_taddr(c);
		}
		sid : string;
		if (c.session != nil) sid = c.session.sid;
		else sid = callid2sid(callid);
		data += "v=0\r\no=- "+sid+" "+sid+" IN IP4 "+daddr+"\r\n";
#		data += "v=0\r\no=username "+sid+" "+sid+" IN IP4 "+daddr+"\r\n";
#		data += "v=0\r\no=username 0 0 IN IP4 "+daddr+"\r\n";
		data += "s=Inferno Ephone Session\r\n";
#		data += "s=\r\n";
		data += "c=IN IP4 "+daddr+"\r\nt="+string rtime()+" 0\r\nm=audio "+string rtpport+
#		data += "c=IN IP4 "+daddr+"\r\nt=0 0\r\nm=audio "+string rtpport+" "+Aproto+" 0\r\
		csp = c.addsession(sid, data);
	}
	msg := header+string len (array of byte data)+"\r\n\r\n"+data;

	if (c.conn == nil) {
		(nil, vaddr, vport, net) := expandnet(proxy(viahost(c, c.tu, 0)));
		if (Dbg) sys->print(Mod+": reconnect to %s at %s!%s!%s\n", vport, net, vaddr, vpor
		(ok, conn) := rdial(net, vaddr, vport, localport(client, vport, vaddr));
		if (ok >= 0) {c.conn = ref conn; c.listen(client);}
	}
```

```
        if (c.conn != nil) {
                if (c.state == "ACK" && !rcode) {
                        if (c.addedsessionp()) c.session.startaudio(0);
                        if (c.session == nil)
                                sys->fprint(stderr, Mod+": missing audio session in %s\n", c.calli
                        else {
                                c.session.announceaudio();
                                call : ref Call; call.switchau(c);
                                c.session.dialaudio();
                        }
                }
                if (csp) {
                        c.session.startaudio(1);
                        c.session.announceaudio();
                }
                if (Vbs > 1) sys->print(Mod+": sending: \r\n%s\r\n", msg);
                fd := c.conn.dfd;
                n := sys->seek(fd, 0, Sys->SEEKSTART);
                if (n < 0) sys->fprint(stderr, Mod+": seek %d %r\n", n);
                n = sys->fprint(fd, "%s", msg);
                if (n < 0) {
                        sys->fprint(stderr, Mod+": sending %d %r\n", n);
                        c.conn.dfd = nil; c.conn = nil;
                        spawn c.resendmsg(client, msg);
                }
                else {
                        if (Vbs) sys->print(Mod+": sent: %s\r\n", c.state);
                        c.msg = msg;
                        c.status(1);
                }
        }
        else sys->fprint(stderr, Mod+": send error: missing connection\n");
        return c;
}

# derive a taddr for response based on received call data
derive_taddr(c : ref Call) : string
{
        taddr : string;
        (nil, tost) := expand2t(lastel(c.path.via), " \t");
        if (taddr != nil) (taddr, nil) = expand2t(tost, ":");
        if (taddr == nil) {
                tost = sipurlval(c.path.contact);
                if (tost != nil) (nil, taddr, nil) = expand(tost);
        }
        if (taddr == nil)
                (nil, taddr, nil) = expand(c.tu);
        return taddr;
}

# this was needed to work around a vovida problem
add_lport(client : string, flag : int) : string
{
        if (flag) {
                (nil, nil, p, nil) := expandnet(client);
                return explicitport(client, p);
        }
        return client;
}

Call.resend(c : self ref Call, client : string)
{
        if (c.msg != nil)       c.resendmsg(client, c.msg);
        else if (Dbg) sys->print(Mod+": no message to resend\n");
}

Call.resendmsg(c : self ref Call, client : string, msg: string)
{
        if (msg != nil) {
                (nil, vaddr, vport, net) := expandnet(proxy(viahost(c, c.tu, 0)));
                lport : string;
                if (c.conn == nil || c.conn.dfd == nil) {
                        lport = localport(client, vport, vaddr);
                        rmdial(net, vaddr, vport, lport);
                        if (Dbg) sys->print(Mod+": reconnect to %s at %s!%s!%s\n", vport, net, vad
                        (ok, conn) := rdial(net, vaddr, vport, lport);
                        if (ok >= 0) {
                                c.conn = ref conn;
                                fd := c.conn.dfd;
                                c.listen(client);
```

```
                        }
                        else {
                                sys->fprint(stderr, Mod+": cannot resend\n");
                                return;
                        }
                }
                fd := c.conn.dfd;
                n := sys->seek(fd, 0, Sys->SEEKSTART);
                if (n < 0) sys->fprint(stderr, Mod+": seek %d %r\n", n);
                n = sys->fprint(fd, "%s", msg);
                if (n < 0) {
                        sys->fprint(stderr, Mod+": resending %d %r\n", n);
                        rmdial(net, vaddr, vport, lport); c.conn.dfd = nil; c.conn = nil;
                }
                else sys->fprint(stderr, Mod+": %s resent\n", c.state);
        }
}

Call.addsession(c : self ref Call, sid, data : string) : int
{
        if (Dbg > 1) sys->print(Mod+": adding session %s\n", data);
        if (c.session == nil)
                c.session = ref Session(sid, data, nil, nil);
        else {
                s := c.session;
                if (s.sid != nil && sid != nil && s.sid != sid) {
                        sys->fprint(stderr, Mod+": changing session id %s->%s\n", s.sid, sid);
                        s.sid = sid;
                }
                if (s.data == nil) s.data = data;
                else s.rdata = data :: s.rdata;
                return 1;
        }
        return 0;
}

Call.addedsessionp(c : self ref Call) : int
{
        s := c.session;
        return (s != nil && s.data != nil && s.rdata != nil);
}

Session.startaudio(s : self ref Session, tipe : int)
{
        data1 := s.data;
        m1 := retrieve("m=", data1);
        c1 := retrieve("c=", data1);
        data2, m2, c2 : string;
        if (Dbg) sys->print(Mod+": session %s data audio:\n\t%s\n", s.sid, m1);
        if (s.rdata != nil) {
                if (Dbg > 1) sys->print("\trdata audio:\n");
                for(l := s.rdata; l != nil; l = tl l) {
                        data2 = hd l;
                        m2 = retrieve("m=", data2);
                        c2 = retrieve("c=", data2);
                        if (Dbg > 1) sys->print("\t:: %s\n", m2);
                        if (m2 != nil) break;
                }
                if (m2 != nil) {
                        setupaudio(s, tipe, snth(2, c1), snth(1, m1), snth(2, c2), snth(1, m2), da
                }
        }
}

debug := 0;
setupaudio(s : ref Session, tipe : int, faddr, fport, taddr, tport, data1, data2 : string)
{
        if (s.audio != nil) {
                sys->fprint(stderr, Mod+": audio already started\n");
                return;
        }
        rtcp1 := int fport;
        if (!rtcp1) {
                if (tipe) fport = default_rtpport;
                else fport = default_rrtport;
        }
        rtcp2 := int tport;
        if (!rtcp2) {
                if (tipe) tport = default_rrtport;
                else tport = default_rtpport;
```

```
        }
        m1 := retrieve("m=", data1); atype1 := snth(2, m1)+"/"+snth(3, m1);
        m2 := retrieve("m=", data2); atype2 := snth(2, m2)+"/"+snth(3, m2);
        if (start("RTP/AVP", atype1)) rtcp1 = 1 + int fport;
        if (start("RTP/AVP", atype2)) rtcp2 = 1 + int tport;
        if (atype1 != atype2) {
                sys->fprint(stderr, Mod+": SDP audio negotiation fail: missmatched %s and %s\n", a
                if (len atype1 > len atype2) atype2 = atype1;
                else atype1 = atype2;
        }
        if (Dbg) sys->print(Mod+": start %s audio: %d %s:%s %s:%s\n\n", atype1, tipe, faddr, fport
        size := 172;
        if (ua != nil) if (C.soleaup(s)) size = ua_seize(size, data1, data2);
        s.audio = ref Audio(faddr+":"+fport+":"+atype1, taddr+":"+tport+":"+atype2, tipe, nil, nil
}

expandatype(t : string) : (string, string, string, string)
{
        (ap, tp, n) := expand3t(t, "/");
        net := "udp";
        if (tp == "TCP") net = "tcp";
        return (net, ap, tp, n);
}

Session.announceaudio(s : self ref Session)
{
        a := s.audio;
        if (a == nil) return;
        if (a.listen) {
                sys->fprint(stderr, Mod+": audio already announced\n");
                return;
        }
        (faddr, fport, ftype) := expand3t(a.addr1, ":");
        (net1, nil, nil, nil) := expandatype(ftype);
        (taddr, tport, ttype) := expand3t(a.addr2, ":");
        (net2, nil, nil, nil) := expandatype(ttype);
        if (!a.tipe) {
                if (Laddr != faddr) sys->fprint(stderr, Mod+": missmatched announce on %s and not
                (ok, conn) := announce(net1, "*", fport);
                if (ok < 0) {
                        sys->fprint(stderr, Mod+": cannot announce %s\n", fport);
                }
                else {
                        if (net1 == "udp") a.conn1 = ref conn;
                        ch := chan of int;
                        spawn audiolistener(net1, a, conn, ch);
                        <- ch;
                        if (a.rtcp1) {
                                (ok, conn) = announce(net1, "*", string a.rtcp1);
                                if (ok < 0) {
                                        sys->fprint(stderr, Mod+": cannot announce rtcp %d\n", a.r
                                }
                                else a.ccon1 = ref conn;
                        }
                }
        }
        else {
                if (Laddr != taddr) sys->fprint(stderr, Mod+": missmatched announce on %s and not
                (ok, conn) := announce(net2, "*", tport);
                if (ok < 0) {
                        sys->fprint(stderr, Mod+": cannot announce %s\n", tport);
                }
                else {
                        if (net2 == "udp") {
                                a.conn2 = ref conn;
                                if (Dbg) sys->print(Mod+": delaying udp audiolisten...\n");
                        }
                        else {
                                ch := chan of int;
                                spawn audiolistener(net2, a, conn, ch);
                                <- ch;
                        }
                        if (a.rtcp2) {
                                (ok, conn) = announce(net2, "*", string a.rtcp2);
                                if (ok < 0) {
                                        sys->fprint(stderr, Mod+": cannot announce rtcp %d\n", a.r
                                }
                                else a.ccon2 = ref conn;
                        }
                }
```

```
        }
}

# pip -- assumes announced local audio port reused to dial to remote audio port
Twinport := 1;
locaudioport(port, locport : string) : string
{
        if (port == nil) return port;
        if (Twinport && locport != nil) return locport;
        return string (int port + 10000);
}

Session.dialaudio(s : self ref Session)
{
        a := s.audio;
        if (a == nil) return;
        if (a.speak) {
                sys->fprint(stderr, Mod+": audio dial already setup\n");
                return;
        }
        (faddr, fport, ftype) := expand3t(a.addr1, ":");
        (net1, nil, nil, nil) := expandatype(ftype);
        (taddr, tport, ttype) := expand3t(a.addr2, ":");
        (net2, nil, nil, nil) := expandatype(ttype);
        if (!a.tipe) {
                ch := chan of int;
                (ok, conn) := dial(net2, taddr, tport, locaudioport(tport, fport));
                if (ok < 0)  {
                        sys->fprint(stderr, Mod+": cannot dial %s%s\n", taddr, tport);
                }
                else {
                        a.conn2 = ref conn;
                        spawn audiospeak(a, conn.dfd, ch);
                        <- ch;
                        if (a.rtcp2) {
                                (ok, conn) = dial(net2, taddr, string a.rtcp2, locaudioport(string
                                if (ok < 0)  {
                                        sys->fprint(stderr, Mod+": cannot dial %s%d\n", taddr, a.r
                                }
                                else a.ccon2 = ref conn;
                        }
                }
        }
        else {
                ch := chan of int;
                (ok, conn) := dial(net1, faddr, fport, locaudioport(fport, tport));
                if (ok < 0)  {
                        sys->fprint(stderr, Mod+": cannot dial %s%s\n", faddr, fport);
                }
                else {
                        if (net2 == "udp") {
                                if (Dbg) sys->print(Mod+": delayed udp audiolisten now starting...
                                spawn audiolisten(a, a.conn2.dfd, ch);
                                <- ch;
                        }
                        a.conn1 = ref conn;
                        spawn audiospeak(a, conn.dfd, ch);
                        <- ch;

                        if (a.rtcp1) {
                                (ok, conn) = dial(net1, faddr, string a.rtcp1, locaudioport(string
                                if (ok < 0)  {
                                        sys->fprint(stderr, Mod+": cannot dial %s%d\n", taddr, a.r
                                }
                                else a.ccon1 = ref conn;
                        }
                }
        }
}

audiolistener(net : string, a : ref Audio, c : Sys->Connection, ch : chan of int)
{
        if (net == "udp") {
                audiolisten(a, c.dfd, ch);
                return;
        }
        ch <- = a.listen = sys->pctl(0, nil);
        p1 := 0;
        while (a.listen) {
                (ok, nc) := sys->listen(c);
```

```
                if(ok < 0) {
                        sys->fprint(stderr, Mod+": listen: %r\n");
                        a.listen = 0;
                        return;
                }
                buf := array[64] of byte;
                l := sys->open(nc.dir+"/remote", sys->OREAD);
                n := sys->read(l, buf, len buf);
                if(n >= 0)
                        if (Dbg) sys->print(Mod+": new audio (%s): %s %s", Mod, nc.dir, string buf

                nc.dfd = sys->open(nc.dir+"/data", sys->ORDWR);
                if(nc.dfd == nil) {
                        sys->fprint(stderr, Mod+": open: %s: %r\n", nc.dir);
                        a.listen = 0;
                        return;
                }
                if (p1) {
                        kill(p1);
                        if (Dbg) sys->print(Mod+": kill previous audiolisten %d\n", p1);
                }
                a.conn1 = ref nc;
                nch := chan of int;
                spawn audiolisten(a, nc.dfd, nch);
                p1 = a.listen = <- nch;
                # expect only one attempt for now!
                return;
        }
}

audiocatchup(a : ref Audio)
{
        if (a.speak) return;
        c := C.finda(a);
        if (c != nil) {
                sys->fprint(stderr, Mod+": dial audio for %s - failed to receive ACK\n", c.callid)
                call : ref Call; call.switchau(c);
                c.session.dialaudio();
        }
}

# Sync is used for loop back test of ephone
# from an emulation version where ua is nil
Sync : chan of array of byte;

audiolisten(a : ref Audio, fd : ref Sys->FD, ch : chan of int)
{
        ok : int; err : string;
        ch <- = a.listen = sys->pctl(0, nil);
        if (!a.size) {
                sys->fprint(stderr, Mod+ ": null buffer size\n");
                return;
        }
        buf := array[a.size] of byte;
        if (Dbg) sys->print(Mod+": audiolisten start size %d\n", a.size);
        cnt := 0; fnt := 0;
        while(a.listen) {
                (lch, nil) := a.lchs;
                if (lch != nil) <- lch;
                n := sys->read(fd, buf, len buf);
                if (n < 0) return;
                else if (n == 0) continue;
                else if (ua != nil) {
                        if (a.tipe && !a.busy) {
                                if (Dbg) sys->print(Mod+": audio now busy = %d\n", n);
###                             audiocatchup(a);
                                a.busy = n;
                        }
                        (ok, err) = ua->playFrame(buf[0:n]);
                        if (ok < 0) break;
                        else if (Dbg > 1 && cnt++ > 1000) {
                                sys->print(Mod+": playFrame %dk len %d\n", ++fnt, n);
                                cnt = 0;
                        }
                }
                # This is the emu to emu sip client test
                else if (n < 30) sys->print(Mod+": hear: %s\n", string buf[0:n]);
                # This is the ephone to emu loopback test
                else {
                        if (Sync == nil) Sync = chan of array of byte;
```

```
                        Sync <- = buf[0:n];
                        if (Dbg && cnt++ > 1000) {
                                sys->print(Mod+": buf len %d\n", n);
                                cnt = 0;
                        }
                }
        }
        if (Dbg) sys->print(Mod+": audiolisten end\n");
        if (ok < 0) sys->fprint(stderr, Mod+": %s\n", err);
}

# in twinport mode, talkdelay may be set to a number
# of frames (e.g. -td 60) recorded and discarded before starting to speak
Talkdelay := 0;
audiospeak(a : ref Audio, fd : ref Sys->FD, ch : chan of int)
{
        ch <- = a.speak = sys->pctl(0, nil);
        if (!a.size) {
                sys->fprint(stderr, Mod+": null buffer size!\n");
                return;
        }
        buf : array of byte;
        ok := 0; err : string;
        if (ua == nil) {
                (faddr, fport, nil) := expand3t(a.addr1, ":");
                (taddr, tport, nil) := expand3t(a.addr2, ":");
                if (a.tipe) err = faddr+":"+fport;
                else err = taddr+":"+tport;
                buf = array of byte ("test from "+err);
                ok = len buf;
        }
        if (ua != nil) buf = array[a.size] of byte;
        cnt := 0; fnt := 0;
        if (Dbg) sys->print(Mod+": audiospeak start size %d\n", a.size);
        # normally we wait to receive audio from caller first (sync udp ports)
        wait := ua != nil && a.tipe;
        # for twinport !p2p we may talk to a lazy mixer -- must talk first
        # but excel 3.0 sip server sends acks off-order -- need talk delay
        if (wait && Twinport && !a.p2p) wait = Talkdelay;
        while(a.speak) {
                (nil, sch) := a.lchs;
                if (sch != nil) <- sch;
                if (ua != nil) {
                        (ok, err) = ua->recordFrame(buf);
                        if (ok < 0) break;
                        else if (Dbg > 1 && cnt++ > 1000) {
                                cnt = 0;
                                sys->print(Mod+": recordFrame %dk len %d\n", ++fnt, ok);
                        }
                        if (wait && a.busy) wait--;
                }
                # This is the emu to ephone loopback test
                else if (Sync != nil) buf = <- Sync;
                # wait for far end to speak first (proxy issue)
                if (wait) continue;
                n := sys->write(fd, buf, ok);
                if (n < 0) return;
                else if (n == 0) continue;
                # This is the emu to emu sip client test
                else if (n < 30) {
                        sys->print(Mod+": speak: %s\n", string buf[0:n]);
                        sys->sleep(2000);
                }
        }
        if (Dbg) sys->print(Mod+": audiospeak end\n");
        if (ok < 0) sys->fprint(stderr, Mod+": %s\n", err);
}

Session.endaudio(s : self ref Session)
{
        if (s == nil) return;
        a := s.audio;
        if (a != nil) {
                if (Dbg) sys->print(Mod+": stop audio: %d %s %s\n\n", a.tipe, a.addr1, a.addr2);
                pid1 := a.listen;
                pid2 := a.speak;
                (lch, sch) := a.lchs;
                a.lchs = (nil, nil);
                if (lch != nil) lch <- = a.listen;
                if (sch != nil) sch <- = a.speak;
```

```
                a.listen = 0;
                a.speak = 0;
                sys->sleep(200);
                kill(pid1);
                kill(pid2);
                # should not be needed - gc does it
                if (a.conn1 != nil) {a.conn1.dfd = nil; a.conn1 = nil;}
                if (a.conn2 != nil) {a.conn2.dfd = nil; a.conn2 = nil;}
                if (a.ccon1 != nil) {a.ccon1.dfd = nil; a.ccon1 = nil;}
                if (a.ccon2 != nil) {a.ccon2.dfd = nil; a.ccon2 = nil;}
                if (ua != nil) if (C.soleaup(s)) ua_release();
        }
        s.audio = nil;
}

Idkey : con 22e+07;
sid2callid(sid : string) : string
{
        return string (int sid - int Idkey);
}

callid2sid(cid : string) : string
{
        return string (int cid | int Idkey);
}

# Preset the audio connections for rtp/tcp tunnelling

Aconn2 : adt
{
        tcpc1 : Sys->Connection;
        tcpc2 : Sys->Connection;
};

Audio2 : ref Aconn2;

tcpaudio()
{
        if (Aproto == "RTP/TCP" && Audio2 == nil) {
                (nil, tcp1) := announce("tcp", "*", Rtpport);
                (nil, tcp2) := announce("tcp", "*", Rrtport);
                Audio2 = ref Aconn2(tcp1, tcp2);
        }
}

tcpclear()
{
        Audio2 = nil;
}

announce(net, addr, port : string) : (int, Sys->Connection)
{
        if (net == "tcp" && Audio2 != nil) {
                if (Dbg) sys->print(Mod+": tcp mode with Audio2 present port %s\n", port);
                if (port == Rtpport) return (0, Audio2.tcpc1);
                else if (port == Rrtport)       return (0, Audio2.tcpc2);
        }
        ur := net+"!"+addr+"!"+port;
        (ok, conn) := sys->announce(ur);

        if (ok < 0)     {
                sys->fprint(stderr, Mod+": cannot announce at %s %r\n", ur);
                return (ok, conn);
        }

  # open the data file for the connection
        if (net == "udp") {
                conn.dfd = sys->open(conn.dir+"/data", sys->ORDWR);
    #conn.dfd = sys->open(conn.dir+"/data", sys->OREAD);

                if (conn.dfd == nil){
                        sys->fprint(stderr, Mod+": cannot open file %s/data: %r\n", conn.dir);
                        return (-1, conn);
                }
        }
        if (Dbg) sys->print(Mod+": announced %s %s port %s\n", ur, conn.dir, port);
        return (ok, conn);
}

listen(client : string, conn : Sys->Connection, ch : chan of int)
```

```
{
        case Transport {
                "UDP" => {
                        cl := ref Client(client, 0, 0, 0);
                        ch <- = active = sys->pctl(0, nil);
                        cl.listen(ref conn, nil);
                }
                * => listener(client, conn, ch);
        }
}

listener(client : string, c : Sys->Connection, ch : chan of int)
{
        if (Dbg) sys->print(Mod+": start tcp listener\n");
        if (ch != nil) ch <- = active = sys->pctl(0, nil);
        else active = sys->pctl(0, nil);
        cl := ref Client(client, 0, 0, 0);
        while (active) {
                (ok, nc) := sys->listen(c);
                if(ok < 0) {
                        sys->fprint(stderr, Mod+": listen: %r\n");
                        active = 0;
                        continue;
                }
                buf := array[64] of byte;
                l := sys->open(nc.dir+"/remote", sys->OREAD);
                n := sys->read(l, buf, len buf);
                if(n >= 0)
                        if (Dbg) sys->print(Mod+": new request (%s): %s %s", Mod, nc.dir, string b

                nc.dfd = sys->open(nc.dir+"/data", sys->ORDWR);
                if(nc.dfd == nil) {
                        sys->fprint(stderr, Mod+": open: %s: %r\n", nc.dir);
                        active = 0;
                        return;
                }
                cl.active = 0;
#               kill(cl.pid);
                cl = ref Client(client, 0, 0, 0);
                spawn cl.listen(ref nc, nch := chan of int);
                <- nch;
                Clist = cl :: Clist;
        }
}

Client : adt
{
        url : string;
        pid : int;
        active : int;
        time : int;
        listen : fn(cl : self ref Client, conn : ref Sys->Connection, ch : chan of int);
        kill : fn(cl : self ref Client, wait : int);
};

Client.kill(cl : self ref Client, wait : int)
{
        cl.active = 0;
        if (wait) sys->sleep(wait);
        if (cl.pid != 0) kill(cl.pid);
}

Clist : list of ref Client;

Client.listen(cl : self ref Client, conn : ref Sys->Connection, ch : chan of int)
{
        client := cl.url;
        (line, address, port) := expand(client);
        cl.active = cl.pid = sys->pctl(0, nil);
        if (ch != nil) {
                ch <- = cl.pid;
                cl.time = time();
                if (Dbg) sys->print(Mod+": spawn listen process %d\n", cl.pid);
        }
        fd := conn.dfd;
        buf := array[1024] of byte;
        while(active && cl.active) {
                if (cl.time) cl.time = time();
                n := sys->seek(fd, 0, Sys->SEEKSTART);
                if (n < 0) sys->fprint(stderr, Mod+": seek %d %r\n", n);
```

```
		n = sys->read(fd, buf, len buf);
		if (n < 0) {
			sys->fprint(stderr, Mod+": receiving %d %r\n", n);
			cl.active = 0;
			continue;
		}
		if (n > 0) {
			csp := 0;
			if (Vbs > 1) sys->print(Mod+": receiving:\n");
			(hl, data) := decode(string buf[0:n]);
			c := mkcall(hl, data);
			if (Vbs) sys->print(Mod+": received [%s]\n", c.state);
			cp := C.find(c.callid);
			(rmeth, rcode, rreason) := c.stateinfo();
			if (cp != nil) {
				(nil, e) := cp.expire;
				(meth, code, reason) := cp.stateinfo();
				if (rmeth == "*") {
					rmeth = meth;
					c.state = rmeth+" "+string rcode+" "+rreason;
				}
				if (code && (!rcode || code == rcode) && meth == rmeth) {
					# dupplicate request (e.g. invite) resend last response
					if (!e && !rcode) spawn cp.resend(client);
					sys->fprint(stderr, Mod+": dupplicate %s %d %s\n", rmeth, :
					continue;
				}
#				if (e == 0 || time() < e)
				cp.expire = (0, 0);
				if (cp.endp()) {
					if (c.endp() && rcode) {C.rem(cp); cp = nil; C.recv = c;}
				}
				else {
					cp.store(c);
					if (c.session != nil)
						csp = cp.addsession(c.session.sid, c.session.data)
					c = cp;
					if (C.this != nil && C.this.callid != c.callid)
						if (Dbg) sys->print(Mod+": switching to received c
					C.recv = c;
				}
			}
			else if (rmeth == "INVITE") {
					C.recv = c;
					if (!rcode && C.this != nil) {
						if (Multicall) {
							if (Dbg) sys->print(Mod+": switching to ne
						}
						else if (C.this.activep()) {
							c.state = "INVITE 486 Busy Here";
							c.send(client);
							continue;
						}
					}
			}
			else if (c.endp()) C.recv = c;
			else {
				if (rmeth == "REGISTER")
					c.status(0);
				else {
					if (C.recv != nil && C.recv.callid == c.callid) {
						(pmeth, pcode, nil) := C.recv.stateinfo();
						if (pcode >= 400) {C.recv = nil; continue;}
					}
					sys->fprint(stderr, Mod+": unexpected new %s in %s\n", c.s
				}
				continue;
			}
			C.take(c);
			c.status(0);

			if (cp != nil && cp.endp()) {
				c = cp;
				c.nextstate(client);
				c.session.endaudio();
				C.rem(cp); C.rem(c); c = nil;
			}
			else if (C.recv != nil && C.recv.endp()) {
				c = C.recv;
				c.nextstate(client);
```

```
                                c.session.endaudio();
                                C.rem(C.recv); c = nil;
                        }
                        else if (c != nil) {
                                if (c.conn == nil) {
                                        (nil, vaddr, vport, net) := expandnet(proxy(viahost(c, c.f:
                                        if (vaddr == Laddr && vport == port) {
                                                if (Dbg) sys->print(Mod+": will not connect to sel:
                                                C.rem(c);
                                                continue;
                                        }
                                        if (Dbg) sys->print(Mod+": connect back to %s at %s!%s!%s\
                                        (ok, conn) := rdial(net, vaddr, vport, localport(client, v
                                        if (ok >= 0) {c.conn = ref conn; c.listen(client);}
                                        else sys->fprint(stderr, Mod+": connect failed\n");
                                }
#                               C.take(c); C.recv = c;
                                if (c.state == "ACK") {
                                        if (c.addedsessionp()) {
                                                c.audiop2p();
                                                call : ref Call; call.switchau(c);
                                                c.session.dialaudio();
                                        }
                                        else sys->fprint(stderr, Mod+": audio setup is missing\n")
                                }
                                else if (csp) {
                                        if (Dbg) sys->print(Mod+": will start audio next...\n");
                                        Sch <- = ("", 0);
                                }
                                c.nextstate(client);
                        }
                }
        }
        cl.pid = 0;
}

cleanClist(f : int)
{
        r : list of ref Client;
        for (l := Clist; l != nil; l = tl l) {
                cl := hd l;
                if (f || cl.active == 0) {
                        if (cl.pid != 0) kill(cl.pid);
                }
                else r = cl :: r;
        }
        Clist = r;
}

Call.activep(c : self ref Call) : int
{
        if (c == nil) return 0;
        (t, n, m) := c.stateinfo();
        return t != "REGISTER" && !c.endp() && (n >= 0 && n < 300);
}

Call.status(c : self ref Call, n : int)
{
        case n {
                0 => {
                        if (c.path != nil && c.path.contact != nil)
                        status("<- "+c.state+": "+ sipurlval(c.path.contact));
                        else if (c.session != nil && c.session.audio != nil && c.session.audio.tip
                        status("<- "+c.state+": "+c.fname+" "+client_sname(c.frum));
                        else status("<- "+c.state+": "+c.tname+" "+client_sname(c.tu));
                }
                1 => {
                        if (Vbs > 1) status("-> "+c.msg);
                        else status("-> "+c.state+": "+c.tname+" "+client_sname(c.tu));
                }
                * => status("CALL: "+c.callid+" "+c.state);
        }
}

Call.nextstate(c : self ref Call, client : string)
{
        case c.state {
                "INVITE" => {
                        c.state += " 180 Ringing";
                        Sch <- = ("r", -1);
```

```
                        c.send(client);
                }
                "INVITE 180 Ringing" => {
                        if (Dbg > 1) sys->print(Mod+": start audible ring\n");
                        Sch <- = ("w", -1);
                }
                "INVITE 200 OK" => {
                        c.state = "ACK";
                        c.send(client);
                }
                "BYE" or "CANCEL" => {
                        c.state += " 200 OK";
                        Sch <- = ("", 0);
                        c.send(client);
                }
#               "CANCEL" => Sch <- = ("", 0);
                * => {
                        (method, code, reason) := c.stateinfo();
                        if (method == "INVITE" && code >= 300) {
                                ohp := onhook();
                                if (code >= 400) {
                                        if (!ohp) {
                                                if (code == 486) Sch <- = ("b", -1);
                                                else Sch <- = ("x", -1);
                                        }
                                }
                                c.state = "ACK "+string code;
                                c.send(client);
                                # redirect case
                                if (code < 400) {
                                        if (c.path != nil && c.path.contact != nil) {
                                                c.tu = sipurlval(c.path.contact);
                                                connect(c.fname, c.tname, c.frum, c.tu, c);
                                        }
                                        else Sch <- = ("x", -1);
                                }
                        }
                        if (code < 300) {
                                sys->fprint(stderr, Mod+": state %s %d %s\n", method, code, reason
                        }
                        else {
                                if (code >= 400)
                                sys->fprint(stderr, Mod+": error state %s %d %s\n", method, code, :
                                else if (code >= 300)
                                sys->fprint(stderr, Mod+": ignored state %s %d %s\n", method, code
                                c.session.endaudio();
                                C.rem(c);
                        }
                }
        }
}

Path : adt
{
        contact : string;
        via : list of string;
        route : list of string;
        record : list of string;
};

mkpath(l : list of string) : ref Path
{
        contact := nonull(findlval("Contact:" :: "m:" :: nil, l, 0));
        via := nolnull(findlall("Via:" :: "v:" :: nil, l, 0));
        if (Dbg > 1)
                if (via != nil) sys->print(Mod+": len via = %d\n", len via);
                else sys->print(Mod+": via ()\n");
        route := nolnull(findalltk("Route:", ",", l, 0));
        if (route != nil) {
                route = sipurls(route);
                if (Dbg > 1)
                        sys->print(Mod+": route (%s . len %d)\n", hd route, len route);
        }
        record := nolnull(findalltk("Record-Route:", ",", l, 0));
        if (record != nil) {
                if (Dbg > 1)
                        sys->print(Mod+": record-route (%s . len %d)\n", hd record, len record);
                if (route == nil) {
                        recurls := sipurls(record);
                        if (contact == nil || findl(sipurlval(contact), recurls)) route = reverse(:
```

```
                        else route = reverse(mksipurl(contact) :: record);
                }
        }
        return ref Path(contact, via, route, record);
}

mksipurl(s : string) : string
{
        s = trimspace(s);
        if (s == nil) {
                sys->fprint(stderr, Mod+": bad () argument to mksipurl\n");
                return nil;
        }
        if (start ("<", s) || start ("sip:", s)) return s;
        else return "<sip:"+s+">";
}

trimspace(s : string) : string
{
        a := 0; b := len s;
        for(i := 0; i < b; i++)
                if (pos(s[i], " \t\r\n") >= 0) a++;
                else break;
        for(i = b -1; i > a; i--)
                if (pos(s[i], " \t\r\n") >= 0) b = i;
                else break;
        return s[a:b];
}

mkcall(l : list of string, data : string) : ref Call
{
        (nil, ll) := sys->tokenize(hd l, " \t");
        method, state, substate : string;
        code := 0;
        if (ll != nil) {
                method = hd ll;
                if (tl ll != nil) {
                        code = int hd tl ll;
                        for(ll = tl ll; ll != nil; ll = tl ll)
                                substate += " "+ hd ll;
                }
        }
        cseq := nonull(findval("CSeq:", l, 0));
        (nil, ll) = sys->tokenize(cseq, " \t");
        if (ll != nil) {
                if (start("SIP/", method)) {
                        if (tl ll != nil) {
                                method = hd tl ll;
                                if (!cseqmethodp(method)) method = "*";
                        }
                }
                cseq = l2string(ll);
        }
        if (Dbg > 1) sys->print(Mod+": received %s %d\n", method, code);
        if (Dbg > 2) sys->print(Mod+": cseq=%s\n", cseq);

        if (!start(" sip:", substate))
                        state = method + substate;
        else state = method;

        path := mkpath(l);
        fname, tname : string;

        (frum, fag) := split(findlval("From:" :: "f:" :: nil, l, 0), "tag=");
        (fname, frum) = sipurlvals(frum);

        (tu, tag) := split(findlval("To:" :: "t:" :: nil, l, 0), "tag=");
        (tname, tu) = sipurlvals(tu);

        if (code >= 200 && method == "INVITE") {
                if (tag == nil) sys->fprint(stderr, Mod+": missing tag in received %s %s\n", metho
                else if (Dbg > 1) sys->print(Mod+": tag=%s in received %s %s\n", tag, method, subs
        }

        callid := nonull(findlval("Call-ID:" :: "i:" :: nil, l, 0));
        if (callid != nil) {
                (nil, ll) = sys->tokenize(callid, " \t");
                if (ll != nil) callid = hd ll;
        }
        sid : string;
```

```
        if (data != nil) {
                p1 := find("o=", data);
                if (p1 < 0) p1 = 0; else p1 = poss(" \t", data, p1);
                if (p1 < 0) p1 = 0;
                p2 := poso('\n', data, p1); if (p2 < 0) p2 = 0;
                (nil, ll) = sys->tokenize(data[p1:p2], " \t\r\n");
                if (ll != nil) sid = hd ll;
                if (Dbg > 1) sys->print(Mod+": received sid = %s\n", sid);
        }
        s : ref Session;
        if (sid != nil)
                s = ref Session(sid, data, nil, nil);
        return ref Call(nil, path, fname, tname, frum, tu, fag, tag, callid, cseq, state, s, (0, 0
}

split(s, k : string) : (string, string)
{
        r : string;
        p1 := find(k, s);
        if (p1 > 0) {
                r = s[p1+len k:];
                s = s[0: p1];
        }
        if (Dbg > 2) sys->print(Mod+": split() -> (%s, %s)\n", s, r);
        return (s, r);
}

sipurls(l : list of string) : list of string
{
        return reverse(revsipurls(l));
}

revsipurls(l : list of string) : list of string
{
        r : list of string;
        for(; l != nil; l = tl l)
                r = sipurlval(hd l) :: r;
        return r;
}

sipurlval_(s : string) : (string, string)
{
        nm : string;
        su := "<sip:";
        p1 := find(su, s);
        if (p1 < 0) return (nil, nil);
        else {
                nm = trimspace(s[0:p1]);
                # lss 2.4 bug
                pn1 := possnot(":", s, p1);
                if (pn1 > p1) p1 = pn1;
                p1 += len su;
        }
        p2 := poso('>', s, p1);
        if (p2 < 0) p2 = len s;
        else ++p2;
        rs := s[p1:p2];
        if (rs != nil) {
                rl := len rs;
                if (rs[rl -1] == '>') { rl--; rs = rs[0:rl]; }
                else sys->fprint(stderr, Mod+": sipurl missing > at end of: %s\n", rs);
                if (Dbg > 2) sys->print(Mod+": sipurl: %s\n", rs);
        }
        return (nm, rs);
}

sipurlval(s : string) : string
{
        (nil, s) = sipurlvals(s);
        return s;
}

sipurlvals(s : string) : (string, string)
{
        (nm, rs) := sipurlval_(s);
        if(rs != nil) return (nm, rs);
        su := "sip:";
        p1 := find(su, s);
        if (p1 < 0) return (nil, nil);
        else {
```

```
                nm = trimspace(s[0:p1]);
                # lss 2.4 bug
                pn1 := possnot(":", s, p1);
                if (pn1 > p1) p1 = pn1;
                p1 += len su;
        }
# lss 2.4 sip server sends these "sip:some name here@dismay:5060" without <>
#       p2 := poss(", \t", s, p1); if (p2 < 0) p2 = len s;
        p2 := poss(",", s, p1); if (p2 < 0) p2 = len s;
#       rs = s[p1:p2];
        rs = trimspace(s[p1:p2]);
        if (rs != nil) {
                (nil, l) := sys->tokenize(rs, ";");
                if (Dbg > 1) sys->print(Mod+": sipurl: %s\n", hd l);
                return (nm, hd l);
        }
        return (nm, rs);
}

decode(s : string) : (list of string, string)
{
        r : list of string;
        data : string;
        p, pn, n : int = 0;
        if (Vbs > 1) sys->print("[");
        while ((p = poso('\r', s, n)) >= 0 || (pn = poso('\n', s, n)) >= 0) {
                if (pn) p = pn;
                if (p > n) r = s[n:p] :: r;
                sl := nonull(getval("Content-Length:", s[n:p], 0));
                if (sl == nil) sl = nonull(getval("l:", s[n:p], 0));
                nc := '\n';
                if (pn) {
                        nc = '\r';
                        pn = 0;
                }
                if (len s > p+1 && s[p+1] == nc) p++;
                if (Vbs > 1) sys->print("%s", s[n:p+1]);
                n = p = p +1;
                if (sl != nil) {
                        l := int sl;
                        data = s[n:n+l];
                        if (Vbs > 1) sys->print("%s\r\n\r\n", data);
                        break;
                }
        }
        if (Vbs > 1) sys->print("]\r\n");
        return (reverse(r), data);
}

# Excel 3.0 sip server too slow to update connections
# maintain a record of connections to server across calls!

Rdc : adt
{
        conn : Sys->Connection;
        net : string;
        addr : string;
        rport : string;
        port : string;
};

Rdials : list of ref Rdc;
rmdial(net, addr, rport, port : string) : int
{
        r : list of ref Rdc;
        n := 0;
        for (l := Rdials; l != nil; l = tl l)
                if ((e := hd l).addr == addr && e.net == net && e.rport == rport && e.port == port
                else r = e :: r;
        Rdials = r;
        return n;
}

rdial(net, addr, rport, port : string) : (int, Sys->Connection)
{
        ok := 0;
        c : Sys->Connection;
        er : ref Rdc;
        r : list of ref Rdc;
        for (l := Rdials; l != nil; l = tl l)
```

```
                if ((e := hd l).addr == addr && e.net == net && e.rport == rport && e.port == port
                else r = e :: r;
        if (er != nil && er.conn.dfd == nil) er = nil;
        if (er != nil) {
                r = er :: r;
                c = er.conn;
                if (Dbg) sys->print(Mod+": reuse dialed %s:%s/%s %s\n", addr, net, rport, port);
        }
        else {
                (ok, c) = dial(net, addr, rport, port);
                if (ok >= 0) r = ref Rdc(c, net, addr, rport, port) :: r;
        }
        Rdials = r;
        return (ok, c);
}

dial(net, addr, rport, port : string) : (int, Sys->Connection)
{
        if (net != "udp") port = nil;
        (ok, conn ) := sys->dial(net+"!"+addr+"!"+rport, port);
        if (ok < 0) {
                sys->fprint (stderr, Mod+": cannot connect to %s!%s!%s %s\n", net, addr, rport, po:
                return (ok, conn);
        }
        if (Dbg) sys->print(Mod+": new connection to %s!%s!%s %s\n", net, addr, rport, port);
        return(ok, conn);
}

# string and list utils

l2string(ll : list of string) : string
{
        r : string;
        for(;ll != nil; ll = tl ll) {
                r += hd ll; if (tl ll != nil) r += " ";
        }
        return r;
}

lastel(l : list of string) : string
{
        for (; l != nil; l = tl l)
                if (tl l == nil) return hd l;
        if (l != nil) return hd l;
        return nil;
}

snth(n: int, s : string) : string
{
        (nil, l) := sys->tokenize(s, " \t\r\n");
        return nth(n, l);
}

snth_token(n: int, s, t : string) : string
{
        (nil, l) := sys->tokenize(s, t);
        return nth(n, l);
}

nth(n: int, l : list of string) : string
{
        for(i := 0; l != nil; l = tl l) {
                if (i == n) return hd l;
                i++;
        }
        return nil;
}

expand2(s : string) : (string, string)
{
        return expand2t(s, ":");
}

expand2t(s, t : string) : (string, string)
{
        (n, l) := sys->tokenize(s, t);
        if (l != nil)
                if (tl l != nil)
                        return (hd l, hd tl l);
                else return (hd l, nil);
```

```
        return (nil, nil);
}

expand3t(s, t : string) : (string, string, string)
{
        (n, l) := sys->tokenize(s, t);
        if (l != nil)
                if (tl l != nil)
                        if (tl tl l != nil)
                                return (hd l, hd tl l, hd tl tl l);
                        else
                                return (hd l, hd tl l, nil);
                else return (hd l, nil, nil);
        return (nil, nil, nil);
}

retrieve(k, s : string) : string
{
        p := find(k, s);
        if (p >= 0) {
                z := poso('\r', s, p);
                if (z < p) z = poso('\n', s, p);
                if (z < p) z = len s;
                return s[p:z];
        }
        return nil;
}

# blank string are nil
nonull(s : string) : string
{
        if (possnot(" \t", s, 0) < 0) return nil;
        return s;
}

nolnull(l : list of string) : list of string
{
        r : list of string;
        for (; l != nil; l = tl l)
                if (nonull(hd l) != nil) r = hd l :: r;
        return reverse(r);
}

pos(e : int, s : string) : int
{
        for(i := 0; i < len s; i++)
                if (e == s[i]) return i;
        return -1;
}

posnot(e : int, s : string) : int
{
        for(i := 0; i < len s; i++)
                if (e != s[i]) return i;
        return -1;
}

poss(t : string, s : string, o : int) : int
{
        if (o < 0) o = 0;
        for(i := o; i < len s; i++)
                for (j := 0 ; j < len t; j ++)
                        if (t[j] == s[i]) return i;
        return -1;
}

possnot(t : string, s : string, o : int) : int
{
        if (o < 0) o = 0;
        for(i := o; i < len s; i++)
                for (j := 0 ; j < len t; j ++)
                        if (t[j] != s[i]) return i;
        return -1;
}

find(e, s : string) : int
{
        for(i := 0; i < len s - len e; i++) {
                ok := 1;
                for (j := 0; j < len e; j++)
```

```
                        if (e[j] != s[i+j]) {ok = 0; break;}
                if (ok) return i;
        }
        return -1;
}

findval(k : string, l : list of string, mc : int) : string
{
        for(r := ""; l != nil; l = tl l)
                if ((r = getval(k, hd l, mc)) != nil) break;
        return r;
}

findlval(kl : list of string, l : list of string, mc : int) : string
{
        for(r := ""; l != nil; l = tl l)
                if ((r = getlval(kl, hd l, mc)) != nil) break;
        return r;
}

nill() : list of string
{
        return nil;
}

# collect values from each entry in l matching k
findall(k : string, l : list of string, mc : int) : list of string
{
        for(r := nill(); l != nil; l = tl l)
                if ((e := getval(k, hd l, mc)) != nil) r = e :: r;
        return reverse(r);
}

# Like findall but also tokenize each element using t tokens
findalltk(k, t : string, l : list of string, mc : int) : list of string
{
        for(r := nill(); l != nil; l = tl l)
                if ((e := getval(k, hd l, mc)) != nil)
                        if (t != nil) {
                                (nil, el) := sys->tokenize(e, t);
                                r = rappend(el, r);
                        }
                        else r = e :: r;
        return reverse(r);
}

# collect values from each entry in l matching one of the keys in kl
findlall(kl : list of string, l : list of string, mc : int) : list of string
{
        for(r := nill(); l != nil; l = tl l)
                if ((e := getlval(kl, hd l, mc)) != nil) r = e :: r;
        return reverse(r);
}

# extend findlall to also tokenize each element using t tokens
findlalltk(kl : list of string, t : string, l : list of string, mc : int) : list of string
{
        for(r := nill(); l != nil; l = tl l)
                if ((e := getlval(kl, hd l, mc)) != nil)
                        if (t != nil) {
                                (nil, el) := sys->tokenize(e, t);
                                r = rappend(el, r);
                        }
                        else r = e :: r;
        return reverse(r);
}

findl(e : string, l : list of string) : int
{
        for(; l != nil; l = tl l) if (e == hd l) return 1;
        return 0;
}

rappend(r, t : list of string) : list of string
{
  for(; r != nil; r = tl r) t = hd r :: t;
  return t;
}

append(h, t : list of string) : list of string
```

```
{
  for(r := reverse(h); r != nil; r = tl r) t = hd r :: t;
  return t;
}

reverse(l : list of string) : list of string
{
        for(r := nill(); l != nil; l = tl l) r = hd l :: r;
        return r;
}

poso(c : int, s : string, o : int) : int
{
        for(i := o; i < len s; i++)
                if (s[i] == c) return i;
        return -1;
}

start(k, s : string) : int
{
        if (len s >= len k && k == s[0:len k])
                return 1;
        return 0;
}

# mc = 1 to match case
getval(k, s : string, mc : int) :string
{
        if (len s < len k) return nil;
        if (mc) {
                if (k == s[0:len k]) return s[len k:];
        }
        else {
                if (equalp(k, s[0:len k]))      return s[len k:];
        }
        return nil;
}

getlval(kl : list of string, s : string, mc : int) : string
{
        for (; kl != nil; kl = tl kl)
                if ((r := getval(hd kl, s, mc)) != nil) return r;
        return nil;
}

equalp(x, y : string) : int
{
        if (len x != len y) return 0;
        for (i := 0; i < len x; i++)
                if (cupcase(x[i]) != cupcase(y[i])) return 0;
        return 1;
}

cupcase(c : int) : int
{
        if ('a' <= c && c <= 'z') return c + 'A' - 'a';
        else return c;
}

downcase(s : string) : string
{
        for (i := 0; i < len s; i++) {
                c := s[i];
                if ('A' <= c && c <= 'Z') s[i] = c + 'a' - 'A';
        }
        return s;
}

upcase(s : string) : string
{
        for (i := 0; i < len s; i++) {
                c := s[i];
                if ('a' <= c && c <= 'z') s[i] = c + 'A' - 'a';
        }
        return s;
}

# Read list from file

readlist(path : string) : list of string
```

```
{
        (ok, dir) := sys->stat(path);
        if (ok < 0) {
                sys->fprint(stderr, Mod+": stat %s: %r\n", path);
                return nil;
        }
        shfd := sys->open(path, sys->OREAD);
        if (shfd == nil) {
                sys->fprint(stderr, Mod+": open %s: %r\n", path);
                return nil;
        }
        lc := dir.length;
        if (lc == 0) return nil;

        buf := array[lc] of byte;
        m := 0; n := lc;
        while ((n = sys->read(shfd, buf[m:], lc - m)) > 0)
                m += n;
        if (n < 0) {
                sys->fprint(stderr, Mod+": read %s: %r\n", path);
                if (!m) return nil;
        }
        if (Dbg > 4) sys->print(Mod+": buf[%d]=%s\n", m, string buf);
        (nil, r) := sys->tokenize(string buf[0:m], " \t\r\n");
        return r;
}

writelist(path : string, l : list of string)
{
        fd := sys->open(path, Sys->OWRITE|Sys->OTRUNC);
        if (fd == nil)
                fd = sys->create(path, Sys->ORDWR, 8r666);
        if (fd == nil) {
                sys->fprint(stderr, Mod+": %s: %r\n", path);
                return;
        }
        sys->seek(fd, 0, Sys->SEEKSTART);
        for (; l != nil; l = tl l)
                sys->fprint(fd, "%s\n", hd l);
}

# Append to file

fappend(path : string, more : string)
{
        fd := sys->open(path, Sys->OWRITE);
        if (fd == nil)
                fd = sys->create(path, Sys->ORDWR, 8r666);
        if (fd == nil) {
                sys->fprint(stderr, Mod+": %s: %r\n", path);
                return;
        }
        sys->seek(fd, 0, Sys->SEEKEND);
        sys->fprint(fd, "%s\n", more);
}

#######  Shannon ephone specific code  #######

# Keypad access on shannon ephone

Dupdsp : con "dsp2mp_dup";

# Default number of digits collected
# will be set to the length of this SIP phone number
Digcnt := 4;

dialplan(sip_client : string)
{
        if (Dialplan != nil && numberp(Dialplan)) {
                n := int Dialplan;
                if (n > 0) {
                        Digcnt = n;
                        sys->print(Mod+": dial plan set to %d digits\n", Digcnt);
                }
                else sys->fprint(stderr, Mod+": unexpected dial plan %s\n", Dialplan);
        }
        else {
                (user, client) := sipurlvals(sip_client);
                (nil, l) := sys->tokenize(client, "@");
                if (l != nil) {
```

```
                        num := hd l;
                        if (!numberp(num)) sys->fprint(stderr, Mod+": unexpected phone number %s\n
                        else {
                                Digcnt = len num;
                                sys->print(Mod+": dial plan set to %d digits\n", Digcnt);
                        }
                }
        }
}

listenkeys(sip_client : string, ch: chan of int)
{
        ch <- = Epid = sys->pctl(0, nil);

        dialplan(sip_client);

        # Checking for a non existing /dev entry after UCBaudio causes kernel dump!
        fd := sys->open( "/tmp/"+Dupdsp, sys->OREAD );

        # if Watch provides a dupplicate channel - use it first
        # else open the DSP device
        if (fd == nil) {
                fd = sys->open( "/dev/dsp2mp", sys->OREAD );
                if (Dbg) sys->print(Mod+": using /dev/dsp2mp\n");
        }
        else
                if (Dbg) sys->print(Mod+": using /tmp/%s from Watch.\n", Dupdsp);

        if(fd == nil) {
                sys->fprint(stderr, Mod+": cannot open /dev/dsp2mp\n");
                return;
        }

        sfd := sys->open(mp+"/"+sipsrv, Sys->OWRITE);
        if (sfd == nil) {
                sys->fprint(stderr, Mod+": open %s/%s: %r\n", mp, sipsrv);
                return;
        }
        keywatch(fd, sfd);
}

keywatch(fd, sfd : ref Sys->FD)
{
        # See shannon/appl/tel/watch.m

        DSP_KEYPRESS     : con 68;
        # HSET_IN_USE_MSG      : con 'o';
        # HSET_NOT_IN_USE_MSG  : con 'p';
  HSIU : con 'o';
  HSNIU : con 'p';
        SPKIU : con 's';
        SPKNIU : con 't';
        hsiu := 0;
        spkiu := 0;

        buf := array[64] of byte;
        n := 0;
        while (Epid) {
                n = sys->read(fd, buf, len buf);
                if (n <= 0) continue;
                case int buf[0] {
                        HSIU => {
                                hsiu = 1;
                                if (!spkiu) machine(sfd, "a");
                        }
                        SPKIU => {
                                spkiu = 1;
                                if (!hsiu) machine(sfd, "a");
                        }
                        HSNIU => {
                                hsiu = 0;
                                if (!spkiu)     machine(sfd, "z");
                        }
                        SPKNIU => {
                                spkiu = 0;
                                if (!hsiu) machine(sfd, "z");
                        }
                        DSP_KEYPRESS =>
                                if (keydigitp(c := int buf[1])) machine(sfd, sys->sprint("%c", c))
                                else sys->fprint(stderr, Mod+": key pressed %d\n", c);
```

```
                }
        }
        if (Dbg) sys->print(Mod+": listenkeys: keywatch end\n");
}

keydigitp(c : int) : int
{
        return (c >= '0' && c <= '9') || c == '#' || c == '*' || c == 'f';
}

# Shannon ephone sound effect FSM

State : adt
{
        s : string;
        d : string;
        c : int;
        f : string;
};

onhook() : int
{
        if (S != nil) return S.s == "z";
        return 1;
}

# to add a new call on flash
keycall() : int
{
        c := "a";
        if (S == nil) S = ref State(nil, nil, 0, nil);
        # already dialing a call
        if (S.s == c && S.c == Digcnt) return 0;
        S.s = c;
        S.d = nil;
        S.c = Digcnt;
        Sch <- = (c, -1);
        return 1;
}

S : ref State;
machine(fd : ref Sys->FD, c : string)
{
        if (S == nil) S = ref State(nil, nil, 0, nil);
        if (Dbg > 1) sys->print(Mod+": key %s\n", c);
        Sch <- = ("", 0);
        case c {
                "a" => {
                        S.s = c;
                        if (C.recv.activep() || C.this.activep()) {
                                fprints(fd, "A");
                                S.c = 0;
                                S.d = nil;
                                S.s = "ok";
                        }
                        else {
                                status("a");
                                S.c = Digcnt;
                                Sch <- = (c, -1);
                        }
                }
                "#" => {
                        if (S.s == "a") {
                                S.c = Digcnt;
                                if (S.d != nil) {
                                        Sch <- = ("", 0);
                                        fprints(fd, S.s+" "+S.d);
                                        S.s = "invite";
                                }
                                else {
                                        Sch <- = (c, Times);
                                        sys->sleep(100);
                                }
                        }
                }
                "z" => {
                        S.s = "z";
                        S.c = 0;
                        S.d = nil;
                        Sch <- = ("", 0);
```

```
                        fprints(fd, S.s);
                }
                * =>
                        if (S.s == "a") {
                                if (c == "f") {fprints(fd, c); return;}
                                S.d += c;       S.c--; status("DIALING "+S.d);
                                Sch <- = (c, Times);
                                sys->sleep(100);
                                if (!S.c) {
                                        Sch <- = ("", 0);
                                        fprints(fd, S.s+" "+S.d);
                                        S.s = "invite";
                                }
                        }
                        else {
                                Sch <- = ("", 0);
                                fprints(fd, c);
                        }
        }
}

fprints(fd : ref Sys->FD, s : string)
{
        b := array of byte s;
        sys->write(fd, b, len b);
}

# Audio conversation support

ua_seize(size : int, data1, data2 : string) : int
{
        m1 := retrieve("m=", data1); atype1 := snth(2, m1); an1 := snth(3, m1);
        m2 := retrieve("m=", data2); atype2 := snth(2, m2); an2 := snth(3, m2);
        if (start("RTP/", atype1) && start("RTP/", atype2) && an1 == "0" && an2 == "0") {
                # Seize the sound system -- disable all sound effects
                Sch <- = (">", 0);
                # Only case Rch is used: synchronize with sound muted
                <- Rch;
                data := data1;
                rtpmap := snth(1, retrieve("a=rtpmap:0", data));
                if (rtpmap == nil)
                        rtpmap = snth(1, retrieve("a=rtpmap:0", data = data2));
                (nil, ptime) := expand2t(retrieve("a=ptime:", data), ":");
                atype := audiotype(rtpmap, ptime);
                ua->setAudioFormat(atype, 1, 8, 12);
                (ok, reason) := audio2open();
                if (ok < 0) sys->fprint(stderr, Mod+": %s\n", reason);
                else {
                        if (Dbg > 1) sys->print(Mod+": UCBAudio open %s %s format buffer size %d\n
                        size = ok;
                        if (debug) {
                                checkua();
                                looptest(size, 1000);
                        }
                }
        }
        else sys->fprint(stderr, Mod+": cannot negotiate audio %s %s\n", atype1, atype2);
        return size;
}

ua_release()
{
        (ok, reason) := ua->audioClose();
        if (ok < 0) sys->fprint(stderr, Mod+": %s\n", reason);
        else if (Dbg > 1) sys->print(Mod+": UCBAudio closed\n");
        Sch <- = ("<", 0);
}

audiotype(rtpmap, ptime : string) : int
{
        case rtpmap {
                "PCMU/8000" =>
                        case ptime {
                                "10" => return UCBAudio->G711MULAWF10;
                                "15" => return UCBAudio->G711MULAWF15;
                                "20" => return UCBAudio->G711MULAWF20;
                                "25" => return UCBAudio->G711MULAWF25;
                                "30" => return UCBAudio->G711MULAWF30;
                        }
                "PCM/8000" =>
```

```
                        case ptime {
                                "10" => return UCBAudio->PCM8000F10;
                                "15" => return UCBAudio->PCM8000F15;
                                "20" => return UCBAudio->PCM8000F20;
                                "25" => return UCBAudio->PCM8000F25;
                                "30" => return UCBAudio->PCM8000F30;
                        }
                * => sys->fprint(stderr, Mod+": cannot set this audio %s %s\n", rtpmap, ptime);
        }
        return UCBAudio->G711MULAWF20;
}

checkua()
{
  # not sure why info is needed in the ua api!
#       info := ua->AudioFormatInfo(0,0,0,0,0,0,0);
        info := ua->AudioFormatInfo(UCBAudio->G711MULAWF20, 8000, 20, 160, 1, 12, 8);
        (ok, reason) := ua->getAudioParams(ref info);
        sys->print(Mod+": UCBAudio audio params %d %s\n", ok, reason);
        sys->print(Mod+": FormatID\t%d\nSampleRate\t%d\nFrameSize\t%d\nFrameBufSize\t%d\nProtocol\t
        (ok, reason) = ua->getSpeakerVol();
        sys->print(Mod+": UCBAudio speaker volume %d %s\n", ok, reason);
        (ok, reason) = ua->getMicGain();
        sys->print(Mod+": UCBAudio mic gain %d %s\n", ok, reason);
}

looptest(size, max : int)
{
        if (ua == nil) return;
        buf := array[size] of byte;
        ok : int; err : string;
        for(i := 0; i < max; i++) {
                (ok, err) = ua->recordFrame(buf);
                if (ok < 0) break;
                (ok, err) = ua->playFrame(buf);
                if (ok < 0) break;
        }
        if (ok < 0) sys->fprint(stderr, Mod+": error: %s\n", err);
}

# Serialized sound effect processor
# Sch is the only allowed interface channel
# to the sound system above this layer

Sch : chan of (string, int);
Rch : chan of (string, int);
Spid := 0;
sound(ch : chan of int)
{
        Sch = chan of (string, int);
        Rch = chan of (string, int);
        ch <- = Spid = sys->pctl(0, nil);
        mute := 0;
        while (Spid) {
                (c, n) := <- Sch;
                case c {
                        ">" => {mute = 1; stopsound(); Rch <- = (c, n);}
                        "<" => mute = 0;
                        * =>
                                if (mute && c != "r" && c != nil) {
                                        if (Dbg > 1) sys->print(Mod+": sound muted -- (%s, %d)\n",
                                }
                                else {
                                        if (Dbg > 1) sys->print(Mod+": sound received (%s, %d)\n",
                                        startsound(c, n);
                                }
                }
        }
        if (Dbg) sys->print(Mod+": sound process ends\n");
}

Soundir : con "/sounds/";
startsound(c : string, n : int)
{
        stopsound();
        # can ring while audio channel is up
        if (c != "r" && C.multiaup()) return;
        f := Soundir;
        case c {
                "" => return;
```

```
                "a" => f += "dialtoneseg.pcm";
                "b" => f += "busy.pcm";
                "c" or "f" => f += "click.pcm";
                "x" => f += "fastbusy.pcm";
                "r" => f = Ringer;
                "w" => f += "ringback.pcm";
                * =>
                        if (Soundp < 2) return;
                        else if (len c == 1 && keydigitp(int c[0]))
                                f += "dtmf"+c+".pcm";
                        else f += c;
        }
        if (Dbg > 1) sys->print(Mod+": f=%s\n", f);
        S.f = f;
        play(f :: string n :: nil);
}

stopsound()
{
        if (S == nil) S = ref State(nil, nil, 0, nil);
        f := S.f;
        S.f = nil;
        if (f != nil) stop(f :: "waitstop" :: nil);
}

killsound()
{
        pid := Spid;
        spawn sendSch("", 0, ch := chan of int);
        killer := <- ch;
        Spid = 0;
        sys->sleep(300);
        kill(pid);
        kill(killer);
}

sendSch(s : string, n : int, ch : chan of int)
{
        if (ch != nil) ch <- = sys->pctl(0, nil);
        Sch <- = (s, n);
}

# Extra ephone and testing testing and debugging

test(args : list of string)
{
        case hd args {
                "d" or "debug" => debug = int hd tl args;
          "p" or "play" or "s" or "stop" =>     {
                        args = tl args;
                        ns := "1";
                        if (tl args != nil) ns = hd tl args;
                        Sch <- = (hd args, int ns);
                }
                "=" or "set" => set(tl args);
                * =>
                        if (hd args != nil && (hd args)[0] == '-') parseopt(args);
                        else usage_internal(hd args);
        }
}

usage_internal(s : string)
{
        if (s != nil) sys->print(Mod+": unknown internal option: %s\n", s);
        sys->print(Mod+": internal options to "+mp+"/"+sipsrv+":\n\td or debug\n\tp or play or s o:
}

Ua : UCBAudio;
set(l : list of string)
{
        if (len l < 2) return;
        case hd l {
                "audio" => Default_audio = tl l;
                "sound" => Soundp = int hd tl l;
                "times" => Times = int hd tl l;
                "timeout" => timeout = int hd tl l; Toa = nil;
                "Timeout" => Timeout = int hd tl l; Toa = nil;
                "ua" => {
                        if (hd tl l == "nil") {
                                if (Ua == nil) Ua = ua; ua = nil;
```

```
                        }
                        else if (Ua == nil) {
                                Ua = ua; ua = Ua;
                        }
                }
                * => return;
        }
        eq : string;
        if (hd l == "audio") eq = sys->sprint("%s = %s", hd l, ls(tl l));
        else eq = sys->sprint("%s = %s\n", hd l, hd tl l);
        sys->print(Mod+": %s", eq);
        status("SET "+eq);
}

ls(l : list of string) : string
{
        r := "(";
        s := " ";
        for (; l != nil; l = tl l) {
                r += hd l;
                if (tl l != nil) r += " ";
        }
        return r + ")";
}

Default_audio : list of string;
Times := 1;

play(args : list of string)
{
        if (ua == nil) return;
        if (args == nil) return;

        f := hd args;
        times := Times;
        args = tl args;
        if (args != nil) {times = int hd args; args = tl args;}

        if (f == Ringer) {
                ch := chan of int;
                spawn ringing(soundcache(f, nil), times, ch);
                <- ch;
                return;
        }
        (typ, proto, jitter, header) := audioinfo();
        if (typ == 0) {
                (name, ext) := expand2t(f, ".");
                typ = audioformat(ext);
        }
        if (typ != 0) {
                ch := chan of int;
                spawn playsound(soundcache(f, typ :: proto :: jitter :: header :: nil), times, ch)
                <- ch;
        }
        else sys->fprint(stderr, Mod+"+ cannot play sample of type %d\n", typ);
}

stop(args : list of string)
{
        if (ua == nil) return;
        if (args == nil) return;
        s := soundcache(hd args, nil);
        if (s == nil) sys->fprint(stderr, Mod+": sound not found %s\n", hd args);
        else if (s.state == 0) sys->fprint(stderr, Mod+": not playing %s\n", s.name);
        else {
                audiop := s.name != Ringer;
                if (tl args != nil) {
                        pid := s.state;
                        s.state = 0;
                        if (Soundp >= 0)
                                timeoutkill(pid, 250, 10, audiop);
                }
                else {
                        if (Soundp >= 0)
                                spawn timeoutkill(s.state, 1500, 200, audiop);
                        s.state = 0;
                }
        }
}
```

```
timeoutkill(pid, tout, quantum, audiop : int)
{
        pstat := "/prog/"+string pid+"/status";
        nc := tout/quantum;
        while(sys->open(pstat, sys->OREAD) != nil) {
                sys->sleep(quantum);
                if (nc-- <= 0) {
                        if (Dbg > 1) sys->print(Mod+": timeout killing %d\n", pid);
                        kill(pid);
                        if (audiop && ua != nil) ua->audioClose();
                        return;
                }
        }
        if (Dbg > 1) sys->print(Mod+": process %d is done\n", pid);
}

Sound : adt
{
        buf : array of byte;
        name : string;
        state : int;
        info : list of int;
};

Cache : list of ref Sound;

soundcache(f : string, info : list of int) : ref Sound
{
        buf : array of string;
        for(l := Cache; l != nil; l = tl l)
                if ((hd l).name == f) return (hd l);
        # Artificial reference to Ringer
        if (f == Ringer) {
                Cache = (s := ref Sound(nil, Ringer, 0, info)) :: Cache;
                return s;
        }
        if (info == nil) return nil;
        (ok, dir) := sys->stat(f);
        if (ok < 0) {
                sys->fprint(stderr, Mod+": stat %s: %r\n", f);
                return nil;
        }
        else {
                fd := sys->open(f, Sys->OREAD);
                n := dir.length;
                buf := array[n] of byte;
                n = sys->read(fd, buf, n);
                if (n < 0) {
                        sys->fprint(stderr, Mod+": read %s: %r\n", f);
                        return nil;
                }
                if (n != dir.length) buf = buf[0:n];
                Cache = (s := ref Sound(buf, f, 0, info)) :: Cache;
                return s;
        }
}

audio2open() : (int, string)
{
        (ok, reason) := ua->audioOpen();
        if (ok < 0) {
                sys->fprint(stderr, Mod+": %s\n", reason);
                (ok, reason) = ua->audioClose();
                if (ok < 0) sys->fprint(stderr, Mod+": %s\n", reason);
                (ok, reason) = ua->audioOpen();
        }
        return (ok, reason);
}

Soundp := 1;
playsound(s : ref Sound, times : int, ch : chan of int)
{
        pid := sys->pctl(0, nil);
        if (s == nil) {
                sys->fprint(stderr, Mod+": sound sample not found\n");
                return;
        }
        buf := s.buf;
        n := len buf;
        info := s.info;
```

```
        (typ, proto, jitter, header) := values4(info);

        if (Dbg > 1) sys->print(Mod+": process %d open %s - %d %d %d %d\n", pid, s.name, typ, proto

        if (Soundp < 0) {ch <- = 0; return;}
        ua->setAudioFormat(typ, proto, jitter, header);
        (ok, reason) := audio2open();
        if (ok < 0) {ch <- = 0; sys->fprint(stderr, Mod+": %s\n", reason);}
        else {
                fs := ok;
                if (Dbg > 1) sys->print(Mod+": playsound %s size %d nframes %d times %d - %d %d %d
                ch <- = s.state = pid;
                while (Soundp > 0 && times--) {
                        for(i := 0; i < n - fs; i += fs)
                                if (!s.state) break;
                                else {
                                        (ok, reason) = ua->playFrame(buf[i:i+fs]);
                                        if (ok < 0) break;
                                }
                        sys->sleep(100);
                }
                if (ok < 0) sys->fprint(stderr, Mod+": %s\n", reason);
        }
        (ok, reason) = ua->audioClose();
        if (ok < 0) sys->fprint(stderr, Mod+": %s\n", reason);
}

values4(l : list of int) : (int, int, int, int)
{
        if (len l == 4) return (hd l, hd tl l, hd tl tl l, hd tl tl tl l);
        return (0, 0, 0, 0);
}

audioinfo() : (int, int, int, int)
{
        l := Default_audio;
        if (len l == 3) return (0, int hd l, int hd tl l, int hd tl tl l);
        if (len l > 3) return (audioformat(hd l), int hd tl l, int hd tl tl l, int hd tl tl tl l);
        return (0, 0, 0, 0);
}

audioformat(ext : string) : int
{
        case ext {
                "pcm" => return UCBAudio->PCM8000F30;
                "pcm20" => return UCBAudio->PCM8000F20;
                "pcm10" => return UCBAudio->PCM8000F10;
                "ulw" => return UCBAudio->G711MULAWF30;
                "ulw20" => return UCBAudio->G711MULAWF20;
                "ulw10" => return UCBAudio->G711MULAWF10;
        }
        return UCBAudio->G711MULAWF20;
}

Ringer : con "ringer";
ringing(s : ref Sound, times : int, ch : chan of int)
{
        if (s == nil) return;
        ch <- = s.state = sys->pctl(0, nil);
        fd := sys->open( "/dev/touch2dsp", sys->OWRITE);
        while(times-- && s.state)
                if (sys->write(fd, array of byte Ringer, len Ringer) <= 0) {
                        sys->fprint(stderr, Mod+": cannot ring this phone\n");
                        return;
                }
                else
                        for(i := 0; i < 20 && s.state ; i++)
                                sys->sleep(200);
}
```